Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Dissertação de Mestrado em Engenharia Informática

1º Semestre, 2008/2009

Integrating the Theme Approach with Aspectual Scenarios

Ana Sofia Conceição Penim

Nº 26222

Orientador

Prof. Doutor João Araújo Júnior

Lisboa

18 de Fevereiro de 2009

Nº do aluno: 26222
Nome: Ana Sofia Conceição Penim

Título da Dissertação: Integrating the Theme Approach with Aspectual Scenarios

Palavras-Chave:

- Desenvolvimento de software orientado a aspectos
- Desenvolvimento baseado em cenários
- Abordagem Theme
- UML
- Cenários negativos

Keywords:

- Aspect-oriented software development
- Scenario based development
- Theme approach
- UML
- Negative Scenarios

Acknowledgments

I would like to thank many people for the help and support through the last two semesters.

First, I would like to express my gratitude to my supervisor, Prof. Dr. João Araújo, for having accepted guiding me through the last year, for helpful discussions, for sharing his knowledge and encouraging me to move forward. Also, I would like to thank Prof. Dra. Fernanda Barbosa and Prof. Dr. Pedro Barahona for approving the first part of this work and allowing me to proceed.

Secondly, to all the friends that have accompanied me: to Manuel Pimenta, for the time spent listening to me, for all the ideas and for all the times that he read and criticized previous versions; to Hélio Dolores, for his friendship, despite not seeing each other. To all my other closest friends: Mário, Pablo, Ana, Andreia, João, Albat. I will never forget you.

Finally, I would like to thank my parents Fernando and Lurdes for all the support, care and love throughout my whole life: the person I am today I owe it to you; and also to my grandmother "Milita" - you are always in my heart and I know I can always count on you.

Thank you all.

Obrigada a todos.

Resumo

A área de Engenharia de Requisitos orientada a aspectos lida com requisitos "crosscutting", isto é, requisitos que estão dispersos pelo documento de requisitos e interligados com outros.
Existem diversas abordagens de requisitos orientadas a aspectos - Theme, proposta por Baniassad e Clarke [7] é uma delas. Esta abordagem é caracterizada pela identificação de um conjunto de acções associadas a verbos presentes nos documentos de requisitos. Estas acções são posteriormente analisadas a fim de encontrar comportamentos "crosscutting", cada uma delas constituindo potencialmente um Theme. O problema desta abordagem é a falta de expressividade do seu mecanismo de composição, mesmo quando os modelos da abordagem são integrados com modelos de análise (como por exemplo, diagramas UML).
A abordagem MATA [24] disponibiliza mecanismos de composição poderosos, baseados em transformação de grafos que usam modelos UML, em particular modelos comportamentais (como por exemplo, diagramas de sequência ou actividade). Estes modelos expressam cenários que constituem uma técnica bastante popular e usada para especificar o comportamento de um sistema. Portanto, o resultado da integração destas duas abordagens será sinergético.
No entanto, e porque num sistema não acontecem apenas as situações esperadas, os cenários podem também ser utilizados para ilustrar situações indesejadas, possibilitando assim o seu tratamento. Consideram-se então também cenários negativos, além dos habituais positivos: a sua representação é idêntica, diferindo apenas na visão optimista, e geralmente assumida, do comportamento do sistema. Estes cenários podem ser identificados com Theme e mapeados para MATA.
Concluindo, esta dissertação tem tem dois principais objectivos: primeiro, a integração das abordagens de Theme e cenários aspectuais (especificados em MATA); segundo, a extensão da abordagem Theme de modo a incluir a modelação de cenários negativos. O resultado passa então pela sinergia de técnicas complementares, incluindo nestas a especificação de situações inesperadas, onde é integrada a modelação de aspectos comportamentais e estruturais.

Abstract

Aspect-oriented requirements engineering emerged to deal with crosscutting requirements, i.e. requirements that are scattered in the requirements document and tangled with other requirements.

There are several aspect-oriented requirements approaches - Theme, proposed by Baniassad and Clarke [7], is one of them. This approach is characterized by the identification of a set of actions associated to verbs present in requirements documentation. These actions are then analyzed in order to identify crosscutting behaviours, each one constituting a potential theme. One problem with this approach is that the composition mechanism is not expressive enough even when the Theme models are integrated to analysis models (e.g. UML diagrams).

The MATA approach [24] provides powerful composition mechanisms, based on graph transformations that used UML models, in particular behaviour models (e.g. sequence or activity like diagrams). These models express scenarios that constitute a very popular and used technique to specify a system's behaviour. Therefore, the result of the integration of these two approaches will be synergetic.

Also, considering that in a system not only the expected situations happen, scenarios can also be used to illustrate unexpected situations, making their treatment possible. Negative scenarios are thus also considered, besides the positive ones: their representation is similar, only differing from an optimist and mostly assumed vision of the system's behavior. These scenarios could be identified with Theme and mapped into MATA.

In summary, the objective of this dissertation is twofold: firstly, we will integrate Theme with Aspectual Scenarios (specified in MATA); secondly, we will extend Theme to include the modeling of negative scenarios. The result will be the synergy between two complementary techniques, including the specification of undesirable situations, where behavioral and structural aspect modeling are integrated.

Acronyms

Below we have a list of acronyms used in this thesis.

ACRONYM	EXPANSION
AOP	Aspect-Oriented Programming
UML	Unified Modeling Language
MATA	Modeling Aspects using a Transformation Approach
AORE	Aspect-Oriented Requirements Engineering
AOSD	Aspect-Oriented Software Development
AOCRE	Aspect-Oriented Component Requirements Engineering
XML	Extensible Markup Language
AORA	Aspect-Oriented Requirements Analysis
MD-AORE	Multi-Dimensional Aspect-Oriented Requirements Engineering

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1
Introduction

With time, hardware and software complexity has increased, leading to maintenance problems. To solve this issue a new paradigm, which attempted to address it by emphasizing units of logic and re-usability in software, has aroused: the object-oriented one. In that approach, each object could be viewed as an independent unit with a distinct role or responsibility.

Thus, the object-oriented paradigm is one of the most important contributions to software development in its history, being based on powerful concepts, including encapsulation, modularity, polymorphism, and inheritance.

However, it is not always possible to modularize all the properties of a certain system using object-orientation. That is the case of *crosscutting concerns*, leading to pieces of code repeated in different places and making it difficult to maintain and to evolve. Also, from a reuse perspective, modules can contain code relating to many concerns. The phenomenon of having code for a concern scattered across multiple parts of the system is known as ***scattering***. When, in turn, multiple concerns are intermixed in the code, this is called ***tangling*** **[**7**].**

Aspect-oriented programming (AOP) [14] was introduced to provide a solution to the scattering and tangling problems, liberating developers from the "hegemony of the dominant decomposition".

In the object-oriented paradigm, the dominant decomposition refers to the modularity of classes and methods, whereas the hegemony refers to the set of decisions that a developer is forced to make and that lead to scattering and tangling, typically when considering non functional requirements such as logging, security, safety, persistence, performance, or availability.

Therefore, aspect-oriented programming attempts to aid programmers in the modularization of crosscutting concerns, which can be seen as a goal, feature, concept, non-functional property or "kind" of functionality [7].

Separation of concerns entails breaking down a program into distinct parts that overlap in functionality as little as possible [7]. Crosscutting concerns are concerns that "cut" across multiple modules in a program, that is, that do not get properly encapsulated in their own modules. This increases the system’s complexity and makes evolution more difficult.

Once again, aspect-oriented programming attempts to solve this problem by allowing the programmer to express crosscutting concerns in standalone modules called aspects, which allow a developer to specify behavior that overlays an existing class model. An aspect can then be defined as a concern whose functionality is triggered by other concerns.

Aspects are used in requirements as well, and they define their own models or are normally integrated into other approaches. In the former case we have Theme [7], which integrates its models with other models such as UML models (e.g. Theme/UML). Some other approaches already integrate aspects into existing models such as viewpoints [18], use cases [13] or scenario-based approaches [25]. A scenario is used to describe a system's behavior regarding its response to outside requests from users. A complete set of scenarios specifies all the possible behaviors of a system, bounding its scope. Scenarios are normally used in the context of use case driven approaches.

A scenario may be seen as an instance of a use case, representing a single path through the use case, being this path the main flow or one of the possible variations, such as options, error conditions, etc. However, a system is not only defined by all the "positive situations": it is possible, and very probable, that some unexpected situation may happen, such as threats or viruses, thus it is desirable for those situations to also be considered during the system's requirements analysis. By the identification of these "negative scenarios", one may also specify how the system should react in order to deal with this kind of situation.

However, their applicability is not limited to object-oriented systems: they can also be modelled to represent crosscutting behavior, i.e., aspects.

1.1 *Motivation and Objectives*

In this dissertation, Theme was chosen as it is one of the most well-known aspect-oriented requirements and analysis approach [7]. It provides a simple mechanism to organize information pertaining to a system's requirements. Theme can be seen as an action performed by the system.
All the possible actions are identified via analysis of the requirements documentation, more precisely by finding all the action words, i.e. verbs, presented in it, being then analyzed in order to recognize crosscutting behavior, i.e., aspects. To compose the base behavior with the aspectual one,

the Theme approach considers the use of templates, and Aspect-J like composition operators. However, these mechanisms provided lack of expressiveness as they cannot express parallel or conditional composition in a simple way. To achieve a more expressive composition, a technique based on UML named MATA (Modeling Aspects using a Transformation Approach) was considered [24].

In order to represent an aspect and the way it affects the system, two sequence diagrams are specified: one showing the base behavior, i.e., the behavior that is constrained by the aspect and the second one depicting the aspect itself. The composition of these two behaviors is achieved by means of graph transformations that specified where, in the base, the aspect should occur.

By integrating the Theme approach with MATA, the previously referred lacks of the first are surpassed, resulting in a more intuitive and comprehensible vision of the system. MATA emerges as an obvious choice of a composition mechanism at scenario level, considering the use of use cases and corresponding sequence diagrams, thus composing the base of the approach itself. Furthermore, and due to the large acceptance of UML in the community, the whole process becomes more clear and intuitive.

Concluding, this dissertation has as objectives:

• To present a study regarding the state-of-the-art of the several approaches related to this work, namely the ones concerning scenarios, aspects and aspectual scenarios;

• To integrate Themes and scenarios (use cases and negative use cases - "misuse cases") to represent themes and "misthemes", which are also considered to enhance the system comprehensibility as well as its completeness;

• To describe the above use cases and misuse cases via sequence diagrams, establishing a base for the composition process;

• To compose the aspectual and base behavior defined in terms of sequence diagrams using MATA, thus surpassing the Theme's lack of expressiveness regarding its composition mechanism at scenario level;

• To illustrate the previous steps, applying them to a case study.

1.2 *Organization of the Document*

This document is structured as follows:

- In Chapter 2 a survey of Aspect-Oriented Requirements Engineering (AORE) is presented,
- In Chapter 3 the Theme Approach is described with a certain level of detail, since it constitutes the main approach considered for this dissertation, being decomposed in several sections, each one describing different concepts of the approach.
- Chapter 4 encompasses an overview of the use of scenarios in system development, where the main application of scenarios, negative scenarios, misuse cases and aspectual scenario-based approaches are described.
- Chapter 5 presents the proposed approach, using a toll collection system as auxiliary case study.
- In Chapter 6 the approach is applied on a case study.
- Chapter 7 provides a comparative analysis of both proposed and effectively used approaches.
- Finally, Chapter 8 includes some conclusions regarding this dissertation work.
- Appendix A presents further examples regarding the “Toll Collection System” case study.
- Appendix B provides further examples regarding the “Phone Features” case study.
- Appendix C provides an example regarding the composition mechanism on the Theme Approach.

Chapter 2
Aspect-Oriented Requirements Engineering

This chapter presents a survey on Aspect-Oriented Requirements Engineering (AORE). It is divided into two main parts: first, the main concepts of Aspect-Oriented Software Development are described; secondly, some existing AORE approaches are presented.

2.1 *Main Concepts of Aspect-Oriented Software Development (AOSD)*

A concern can be seen as a system's functionality, as something that the system must take into account, i.e., as a property that the system must provide. An *aspect*, as described in [7], is a particular kind of concern, whose functionality is triggered by other concerns and in multiple situations. Separating concerns into aspects avoids the need to explicitly trigger their functionalities within the code related to other concerns, tangling the two concerns, and also the scattering inherent to triggering in multiple places. These phenomena can be seen in Figure 2-1.

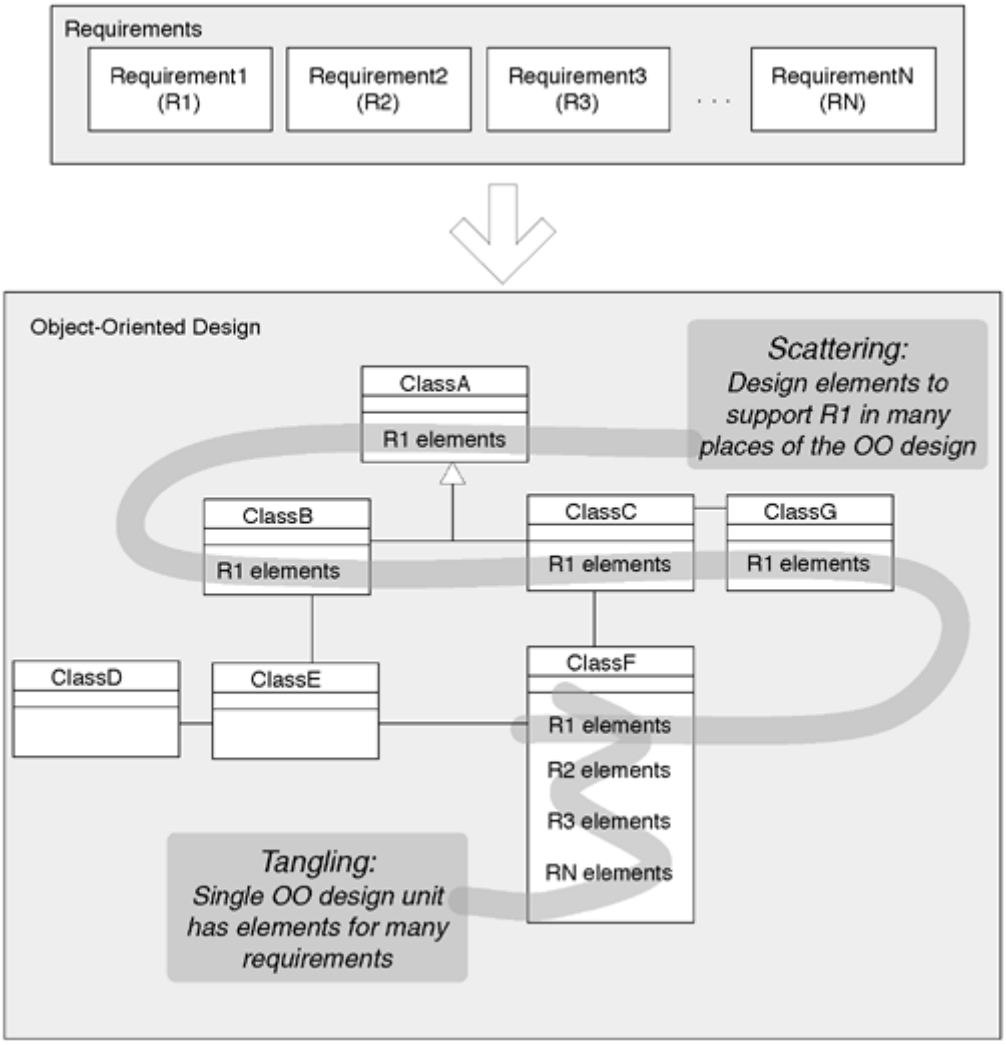

Figure 2-1 Tangling and Scattering

Aspects are a "programmatic construct" and provide active support for separating concerns in source code.

At the beginning of aspect orientation, developers used object-oriented methods and languages (such as standard **UML - *Unified Modeling Language***) for designing their aspects, but UML was not designed to provide constructs to describe aspects. In what concerns designing aspects, there is a need for special design constructs that enable the separation of crosscutting functionalities, seeing as it is the only way to improve the whole process and offer traceability to aspect-oriented code. There are two approaches that stand out in the aspect-oriented language development world: the asymmetric and the symmetric approaches. The **asymmetric approach**, as shown in Figure 2-2, considers aspects as external events or modules, separate from the core of the program, that are triggered when required; they add dynamic behavior to a system and its main functionality, leaving it to retain only the structure and behavior that pertains to the system's domains. Hence, an aspect

will be treated as a highly used operation that is invoked along a core execution thread, seeing as it only justifies its existence if it is triggered more than once.

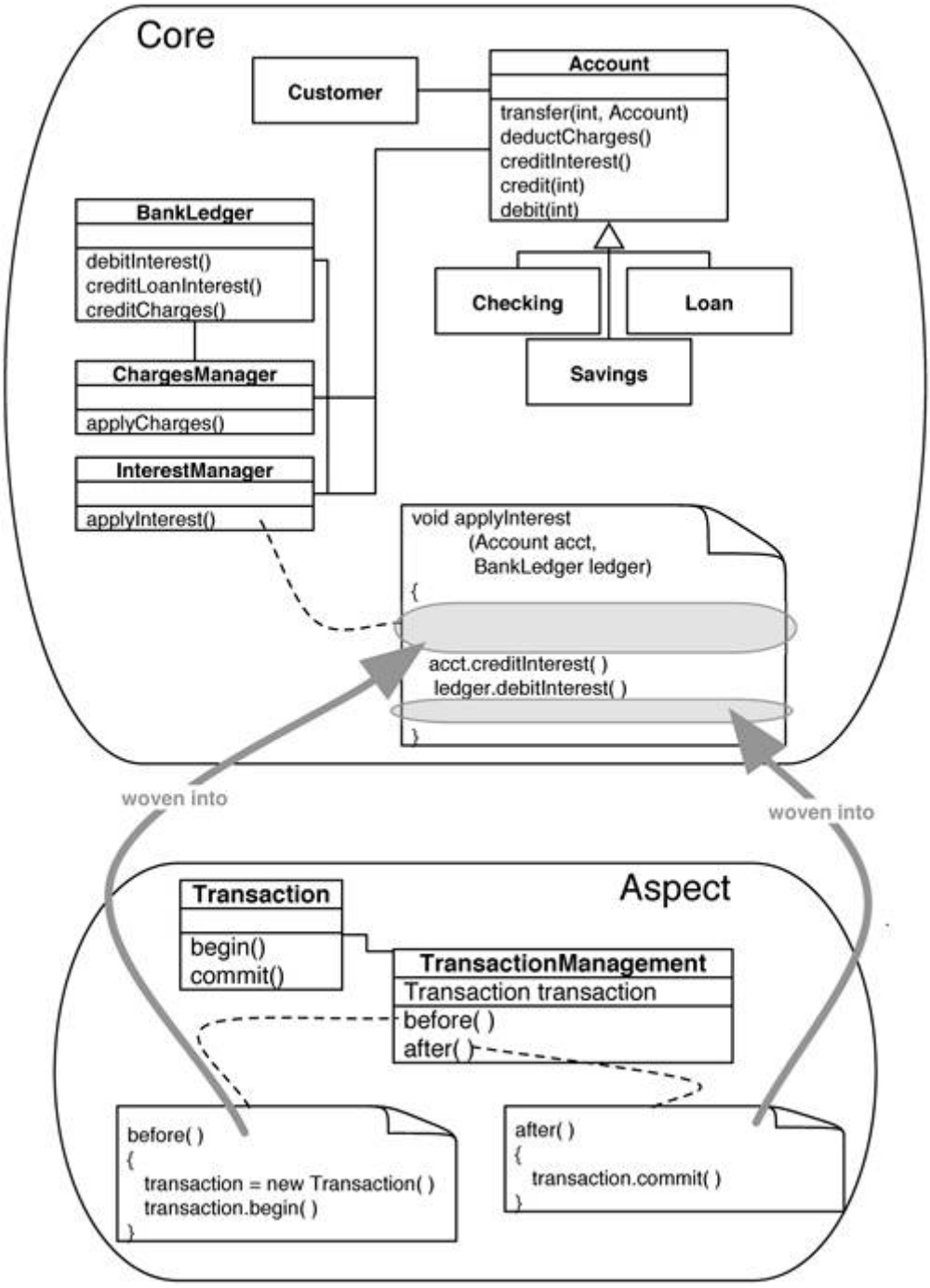

Figure 2-2 Aspects and core in the asymmetric paradigm

In the **symmetric separation model**, depicted in Figure 2-3, besides the modularization of aspects, the core is also analyzed for further modularization. An entire system is therefore made up of bits of

separate functionality that could be thought of as features or concerns, which can then be recombined in various ways to form a functioning whole.

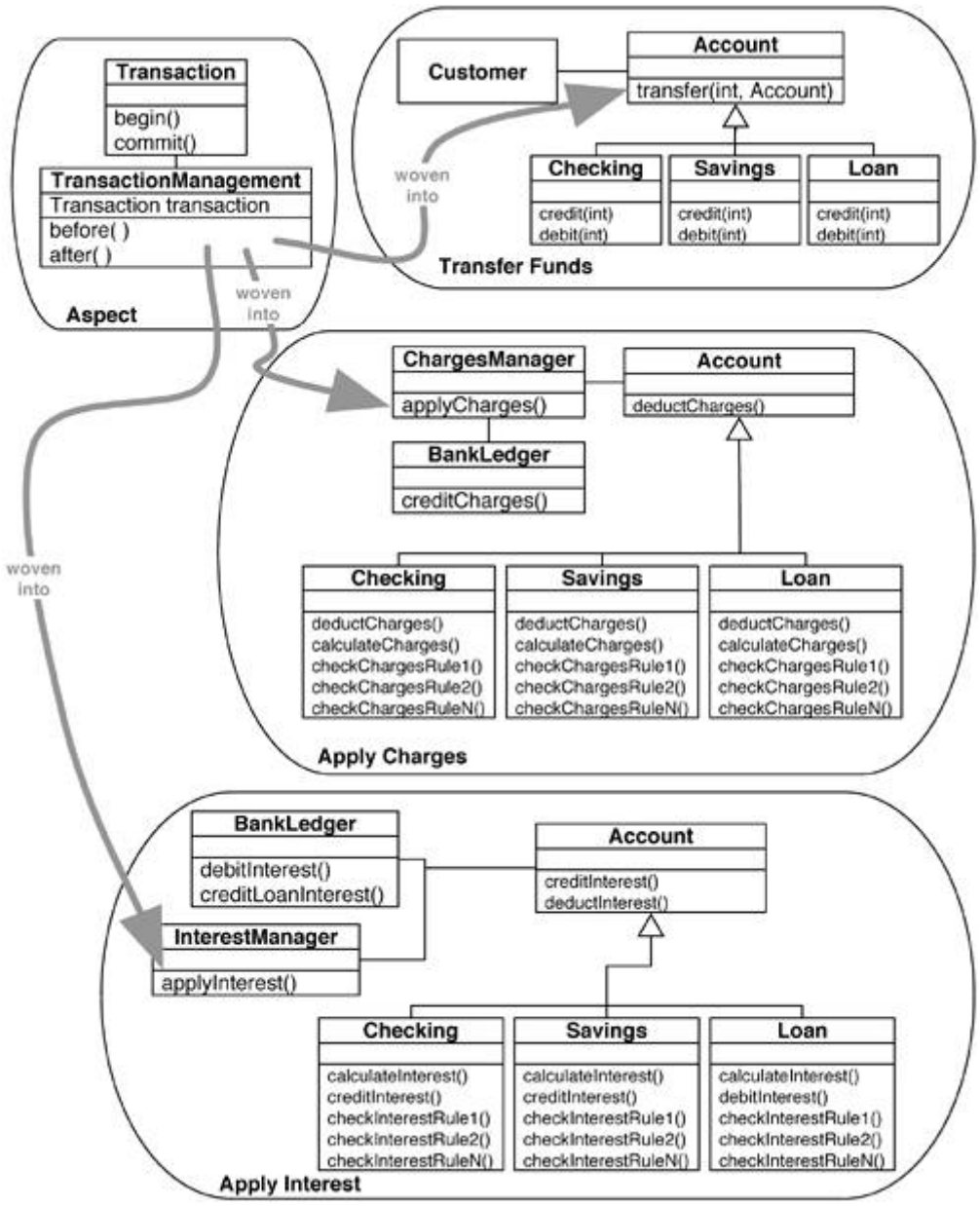

Figure 2-3 Separation of different features in the symmetric paradigm

The duplication described above, despite of looking worse than scattering, is required in order to provide a complete view of the system from the perspective of a particular concern, enhancing the separate understandability of it. This understandability is achieved because only and all relevant functionality for a concern is present within the concern module (*locality*), which also enhances maintainability.

2.2 Aspect-Oriented Requirements Engineering Approaches

Requirements engineering is the process of reasoning about the problem domain and discovering the stakeholders' needs [3]. The understanding of these needs will take the form of a requirements specification that will constitute a connection between the problem domain and the solution domain. Aspect-oriented requirements engineering (AORE) attempts to treat crosscutting properties in a systematic manner, thus facilitating the reasoning about their impact in the problem domain as well as modularizing such properties in the requirements specification in order to map and trace them to the solution domain [3].

The need for an effective separation of concerns has long been recognized by requirements engineering approaches. Viewpoint-oriented techniques [18] consist of partitioning a system's requirements based on subjective perspectives derived from stakeholder's views. Use cases provide descriptions of the system from the perspective of its use by the actors that interact with it.
However, despite the subjective perspectives provided by viewpoints or use cases, they do not treat non-functional properties, which often form good candidates for aspects, in a systematic manner. There are some approaches that do treat them, such as PREView [20], but they only warn the requirements engineer about the possibility of these properties affecting certain viewpoints and do not clearly tell how they constrain or influence specific requirements in the system.

AORE [3] has as an objective to solve these problems by "*providing a systematic means for the identification, modularization, representation and composition of crosscutting properties, both functional and non-functional ones, during requirements engineering*" [3]. AORE techniques consist on providing a fine-grained specification of how a requirements-level aspect constrains or influences specific requirements in a system. By understanding exactly the composition between aspectual and non-aspectual requirements, the understanding of their interaction, inter-relationships and conflicts is enhanced, thus helping to identify trade-offs early on in the development life cycle and undertake develop negotiations with the affected stakeholders.

An aspect may be represented by a single requirement or a coherent set of requirements that may constrain or influence the specified behavior of several other requirements or may influence them by altering their specified behavior.

AORE is characterized by four main characteristics:

- The existence of a process to identify crosscutting properties in a requirements specification, which may be accomplished by providing intuitive guidelines for this purpose, identifying specific keywords or action words in a requirements specification, or using specific tool support based on semantic analysis of requirements documents.
- The capability to modularize crosscutting properties belonging to a particular concern in one requirements-level module.
- The existence of appropriate means to represent requirements-level aspects, which might change depending on the application domain or availability of relevant tools, being it graphical or assuming a semi-structured format.
- The capability to compose, aspectual and non-aspectual requirements in order to understand exactly how requirements-level aspects affect other system requirements.

2.2.1 Aspect-Oriented Component Requirements Engineering (AOCRE)

This approach [11] consists of identifying and specifying the functional and non-functional requirements related to "key aspects" of a system that each component provides or requires. A component may have many or few aspects and an aspect may share component services. These "overlapping" aspects are an obvious consequence of high-level categorization of the systemic properties of components, helping requirements engineers to better understand the related component characteristics. Thus, aspect characterization may be seen as a way to encapsulate multiple and systematic perspectives into components, improving the understanding and reasoning about component data, functionality, constraints and inter-relationships.

The approach begins, as shown in Figure 2-4, with the analysis of general application requirements, individual or groups of components requirements, which allows iterative top-down and bottom-up requirements refinement. The engineers proceed by characterizing the components' aspects and their details, provided and required details, functional and non-functional properties and reasoning about the inter-related components' aggregate aspects. The identified components and aspects are refined into detailed component designs.

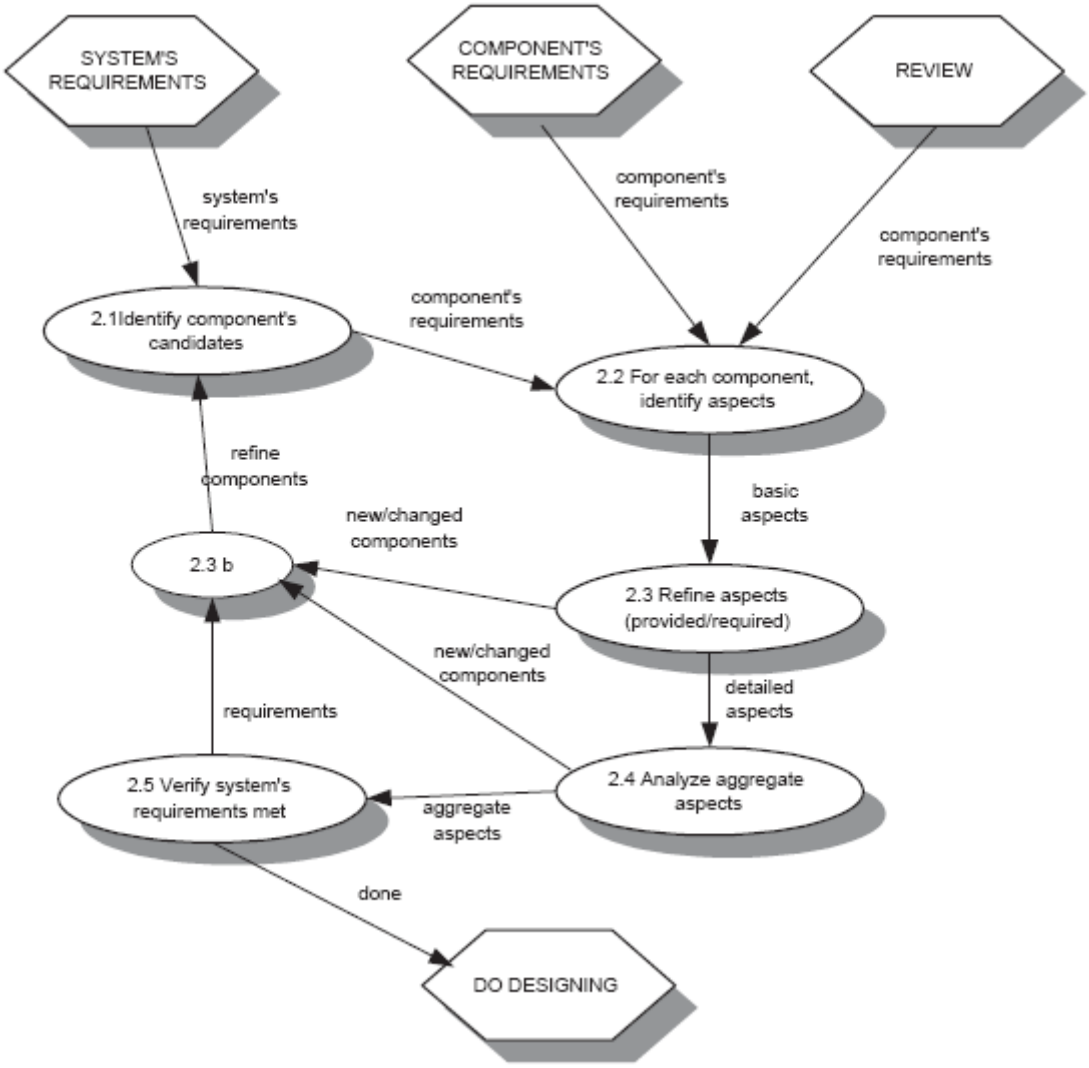

Figure 2-4 The AOCRE process

For each component, possibly from stakeholder requirements and object services, aspects for which the component provides or requires services from other components are identified. After identifying a component's provided and required aspects, inter-component relationships can be analyzed. Aggregate aspects can be identified and specified for groups of interrelated components. This allows Engineers to reason about aspect-oriented requirements for a set of related components. Aspect-oriented component requirements are useful when designing and implementing components, providing a set of functional and non-functional constraints that can refine a design and a specification that can be used to test the implementation. Components at requirements-level can be refined directly to matching software components at design-level or can be split, merged or even revised, like requirements-level component aspects.

2.2.2 Separation of crosscutting concerns from requirements to design via a use case approach

Despite the several different approaches on Aspect-Oriented Software Development, none of those cover the software development from requirements to design. This approach [21], adapting a use case driven approach, aims to explicitly provide the reasoning and separation of crosscutting concerns, including the non-functional ones, from requirements artifacts to design artifacts.

Use cases, as explained in more detail in [21], are used to represent the most important business concepts and events that occur in the system, being those functional or not, showing all the actors and theirs interactions. Then, they are specified in more detail and structured for the next stages: the analysis and design activities which will result in classes and subsystems.

In order to deal with non-functional concerns, the authors of this approach have included in the requirements activities the **NFR Framework**, treating those concerns as goals to be achieved.

The NFR Framework begins with the decomposition of each non-functional concern into more specific ones, which are then operationalized via certain mechanisms. During these stages, contributions and conflicts are analyzed, establishing the impact of the non-functional concerns on each other and identifying priorities. Finally, the operationalizations are accepted or rejected.

Non-functional concerns are normally crosscutting, thus their adequate treatment is an important step in Aspect-Oriented Software Development. In order to separate crosscutting behavior in use cases, the crosscutting behavior will be placed in a separate use case (crosscutting use case) that will be connected to the use cases it affects via a new kind of relationship named crosscuts. The information about the composition between the crosscutting use case and the ones affected by it are described in a composition table.

Each analysis class should represent the specific behavior of an entity that collaborates to fulfill the use cases. The separated entity that encapsulates the crosscutting behavior is represented in a crosscutting class.
Each identified crosscutting requirement should originate a crosscutting class, which, in turn, can represent more than one analysis classes necessary to concretize a crosscutting behavior.

When the analysis classes have been identified and specified, the interactions between each of the use case and their corresponding analysis objects are represented in collaborations diagrams. Separation of crosscutting concerns should also be maintained in the interactions diagrams, thus crosscutting objects that affect analysis objects interactions should not be included in the interaction diagram - the information about how a crosscutting object will affect analysis objects interactions should be described in a composition table.
Aspects are characterized by adding new behavior or structure to classes and are stereotyped with «aspect». The crosscutting behavior will be modeled as operations and new structure as properties in the aspectual class. The composition tables continue to be used in order to determine the join points and the composition rules.
It is possible to determine the classes affected by the aspect by analyzing the composition table of its aspectual class. A "crosscuts" association is used to model the relationship between an aspect and the classes affected by it.

2.2.3 Aspect-Oriented Requirements Engineering (AORE) with ARCaDe

Techniques such as viewpoints and use cases are helpful when dealing with separation of stakeholders' concerns but do not guarantee any consistency between those and global requirements and constraints. The AORE approach, supported by the Aspectual Requirements Composition and Decision support tool (*ARCaDe*), and as described in [19] [18], which is one of the AORE approaches and not the general AORE methodology, aims at resolving these issues by supporting separation of crosscutting functional and non-functional properties at the requirements level, defining composition rules in modules that represent coherent abstractions and follow well-defined templates. The composition rules, including actions and operators, specify how an aspectual requirement influences or constrains the behavior of a set of non-aspectual requirements. With an early modularization, it is possible and easier to establish trade-offs between aspectual requirements, providing support for negotiation and decisions among stakeholders, as well as to determine their mapping and influence on artifacts at later development stages.

This approach's first stage consists of identifying and specifying both concerns and stakeholders' requirements using viewpoints, use cases, goals or problem frames. By relating these concerns and requirements through a matrix it is possible to see which concerns crosscut the modules encapsulating stakeholders' requirements and qualify them as candidate aspects. Having these

relationships defined, the next step consists of defining the composition rules. Once the composition rules have been defined, identification and resolution of conflicts between candidate aspects are accomplished in three steps:

- A contribution matrix showing the positive or negative contribution of each aspect to the other is built;
- Weights are attributed to those aspects that contribute negatively to each other regarding a set of stakeholders' requirements;
- Conflicts are solved with the stakeholders using the previous defined prioritization.

Conflict resolution might originate a revision of the requirements specification. In this case, the requirements are recomposed and possibly new conflicts are resolved. This process is repeated until all conflicts have been resolved.

The stakeholder requirements are identified using viewpoints and specified using XML, which is also used to specify the identified aspects and the composition rules that relate viewpoints with aspects. Viewpoint requirements, aspectual requirements and composition rules are defined using pre-defined templates, being supported by the ARCaDe tool. XML has been chosen due to its capability to be extensible, since it is not possible to anticipate the various types of composition operators and actions that might be required. Viewpoints and concerns are described in a similar way in XML: a *Viewpoint* or *Concern* tag, respectively, denotes the start of a Viewpoint or Concern. Sub-viewpoints and sub-requirements are nested under their correspondent viewpoints or requirements, respectively. Each requirement has a unique *id*, which identifies it in the viewpoint. Viewpoint and Concern names are also unique. After identifying and describing viewpoints and concerns, they are linked using a matrix. Analyzing the previous matrix, a concern is identified as a candidate aspect if it crosscuts several viewpoints. When a candidate aspect is identified, the XML specification of the corresponding *Concern* is transformed to reflect this fact, being the *Concern* tag replaced with an *Aspect* tag.

The relationships between aspectual requirements and viewpoint requirements are defined using composition rules - a *Composition* tag encapsulates a coherent set of composition rules. Here, in opposition to what happens in the previous sections, each *Requirement* tag has at least two attributes: the *aspect* or *viewpoint* it is defined in and a unique *id* which identifies it within its defining scope. Any sub-requirements belonging to a viewpoint requirement must be explicitly excluded or included in the *Constraint* imposed by an aspectual requirement; this is accomplished using an *include* or *exclude* value in the optional *children* attribute. The *Constraint* tag defines an

action and operator defining how the viewpoint requirements are to be constrained by the specific aspectual requirements.

The composition rules are also used to compose aspects and viewpoints, leading to the identification of conflicts among aspects whose requirements constrain the same or overlapping sets of viewpoint requirements. The contribution table shows in what way an aspect contributes to the other, negatively or positively.

The detected conflicts can be resolved by attributing weights to the cells of the aspect vs viewpoint matrix, constructed as the concern/viewpoint matrix previously presented.

2.2.4 Vision and Aspects

Requirements elicitation consists in capturing and organizing all the system information. In this approach [8], the several viewpoints are described using templates, their relationships are illustrated in viewpoint diagrams and their interactions are shown in sequence diagrams.

Each NFR is described using a template in which its name, description and influence on viewpoints and use cases are described. Furthermore, since a NFR may be related with other ones, contributions and priority are also described.

After the requirements elicitation stage, the relevant viewpoints to the system are identified and specified in terms of their name, sources, actors, use cases, NFRs, relationships with other viewpoints and history. A viewpoint can also be represented as a UML class with the stereotype ***«Viewpoint»*.**

Actors and use cases of the system's viewpoints are identified via requirements analysis and are documented in a template containing the use case's name, description, related actors and viewpoints. The specifications of its primary and secondary scenarios are also included, as well as other scenarios related to it via «extends» or «include». Non-functional requirements are also listed. Use cases define the functionalities of a system available to their actors; actors and use cases can be associated to more than one viewpoint.

Having identified NFRs, viewpoints and their use cases, one must check which viewpoints and use cases are influenced by a NFR and for each viewpoint/use case which NFRs affect it. This is done by analyzing the previously described templates.

After the identification and specification of use cases, it is possible to check whether there is a use case included by more than one use case of one or more viewpoints or if there is a use case that extends more than one use case, that is, if any use case crosscuts several use cases, thus being

identified as an aspectual use case. An aspectual use case would crosscut several viewpoints. Viewpoints can also be part of other viewpoints, defining a viewpoint aggregation. If two viewpoints are associated, then there must exist an interaction between the actors of those viewpoints.

The identification and resolution of conflicts are treated the same way as in the AORE approach. For candidate aspects in conflict with the same viewpoints and use cases, weights are attributed to them. The authors also suggest creating two other similar tables to relate aspectual use cases to viewpoints and to relate aspectual use cases to use cases.

2.2.5 Aspect-Oriented Requirements Analysis (AORA)

The idea behind this approach [6] is to provide a contribution for aspect-orientation by integrating non-functional with functional requirements, also taking in account their crosscutting nature and influence over each other. The whole process is constituted by four main tasks, namely "Concerns Identification", "Concerns Specification", "Concerns Modelling in UML" and "Concerns Composition", each one being composed by several sub-tasks.

A concern is a conceptual subject of interest in a system, being possibly defined by a coherent set of requirements, each one defining a property that the future system must provide. Therefore, the first task is responsible for identifying all the concerns of a system, by means of its stakeholders and sources. The identified concerns can then be decomposed into subconcerns. Important information about each concern should be registered in a template that should be filled iteratively and incrementally as the several tasks are applied. In this stage, rows *Name*, *Source*, *Stakeholder* and *Description* must be defined. The remaining rows, *Responsibilities*, *Contributions*, *Priorities* and *Required concerns*, will be filled in the next step. The next step, specification of concerns, consists of four subtasks, each one responsible for identifying responsibilities, contributions between concerns in order to detect possible conflicts, priorities and required concerns, respectively. The empty rows left in the previous step must be filled here. A UML requirements analysis model can be build based on the collected information - for this, an extended use case is used. In order to better characterize the relationships between concerns, new stereotypes are defined:

- **«overlap»:** used when a given concern is required by another concern and may improve its behavior;
- **«bind»:** used when a given concern is required by and constrains the behavior of another concern.

The final task is composed by four subtasks and supports conflict management between concerns, resulting in context-dependent compositions, which are defined by identifying match points, crosscutting concerns, handling conflicts and defining composition rules for each match point. Analyzing the *Required concerns* row, it is possible to identify the (match) points where the composition will take place. A match point for a concern consists of a set of concerns required by the first one. In order to facilitate conflicts resolution, it may be useful to relate each match point to the stakeholders related to it, since the composition rules are defined for each match point. Thus, when a conflict emerges, one can easily verify those of the system's stakeholders that need to be contacted to manage the conflict. Furthermore, the crosscutting concerns can also be identified as the ones that are required by more than one other concern. Based on the *Contribution* row it is possible to check if, for a given match point, any of the involved concerns contribute negatively to any other. A conflicting situation may arise if more than one concern is to be applied in a certain match point and if negative contributions between them are involved. If two concerns contribute negatively to each other and have the same priority one needs to verify the *Priority* row: if each concern has the same priority a trade-off must be negotiated with the stakeholder; if they have different priorities, the dominant concern is the one with the higher priority. The final subtask consists in defining the composition rules that define the order in which the concerns will be applied in a particular match point. These rules are composed by a set of LOTOS [9] based operators.

2.2.6 Multi-Dimensional Separation of Concerns (MD-AORE)

There are several different approaches to Requirements Engineering proposing techniques for the decomposition and composition of concerns; however all of them use a dominant dimension as the base composition, with other dimensions crosscutting them - it is a two-dimensional separation. As what happens with non-functional requirements, the functional ones are also responsible for crosscutting situations. In this mostly used two-dimensional separation, the influence of those functional requirements and their associated trade-offs are simply ignored.

This new model [15] [16] proposes to deal with requirements decomposition in a uniform fashion, i.e. being those functional or non-functional. In this context, and as a key characteristic of the approach, a meta-concern space is defined and the notion of a compositional intersection is introduced. A meta-concern space makes it possible to derive concrete system-specific concerns

based on the specific features of the problem domain. The compositional intersection is responsible for the identification of suitable sets of concerns as a basis to discover trade-offs between them.

In this multi-dimensional approach, concerns imply any coherent set of requirements. A concern space at requirements level is defined as a hypercube, with each of its faces representing a particular concern of interest. By treating all concerns in the same manner, it is possible to choose any set of concerns as a base to project the influence of another concern or set of concerns onto this base. In this proposal, concern identification is based on the observation that certain concerns, both functional and non-functional, appear time and again during system development.

Like in the previously described approaches, well-defined templates based on XML were chosen to represent both the abstract concerns as well as their concrete realizations. The definition of concerns and respective requirements is done in the same manner as in Section 2.2.3.

Having categorised the various requirements of the system into concrete concerns, the next step consists of defining composition rules for each concern. The composition rules are defined in a way similar to that shown in Section 2.2.3. The main difference is that a concern constrains other concerns and not only viewpoints.

In order to identify trade-off points, the interactions of a concern with other concerns with reference to some base are observed. The previously mentioned notion of a composition intersection, denoted by ∩, defines a constraint to the potential combinations of concerns to be used as a base for the combinations of concerns that are involved in trade-offs. Trade-offs are analyzed based on the positive, negative or "none" contribution that one concern may have on another with respect to the base identified via the compositional intersection.

Like in the previous approaches, a contribution matrix will display, on each cell, the type of contribution and also the compositional intersection set used to find the contribution. Empty cells denote the inexistence of a relationship.

After dealing with and analyzing trade-offs, architectures choices may be discussed. Each concern in the multi-dimensional separation leads to a number of architecture choices, which are unlikely to be the same and could even result in conflicts, influencing the final architecture choice. The previous trade-off analysis is also useful in the early resolution of these conflicts via stakeholders negotiations and prioritization of concerns.

2.2 *Summary*

This chapter has described the main concepts of AOSD, such as the notions of aspect and various phenomena related to it, namely scattering and tangling.

Also, besides the description of the AORE process itself, some AORE approaches were briefly explained. These approaches relate aspects to components, use cases or viewpoints.

Having defined the role of aspects in Requirements Engineering and how they may affect a system, the next Chapter introduces another aspect-oriented approach which constitutes the main one under this work.

Chapter 3
The Theme Approach

Themes [7] are more general than aspects and more closely encompass concerns as described for the symmetric approach, since themes are individual concerns regardless of whether they are aspects or separated concerns that would be located in the core. However, the terms crosscutting and aspect are defined as in the asymmetric paradigm: as a functionality that is triggered in multiple situations.

In this Chapter we present the Theme approach by describing its main concepts, the main parts (Theme/Doc and Theme /UML) and its process.

3.1 *Concepts*

A theme is, at a requirements-level, a subset of responsibilities and, at a design level, includes the structure and behavior required to carry them out.

A concern can be seen as an interest related to the development, operation, use or evolution of the system.

Inter-Theme relations are similar to those between aspects or features and the rest of the system; this fact allows for overlapping to occur in two ways: ***concept sharing*** and ***crosscutting***. It is also possible, however rare, for themes to be completely independent.

Different themes may have design elements that represent the same core concepts in the domain. ***Concept sharing*** (Figure 3-1) is one category of crosscutting in the symmetric separation model and is not discussed in the asymmetric one. Encapsulation in this manner has the benefit of locality.

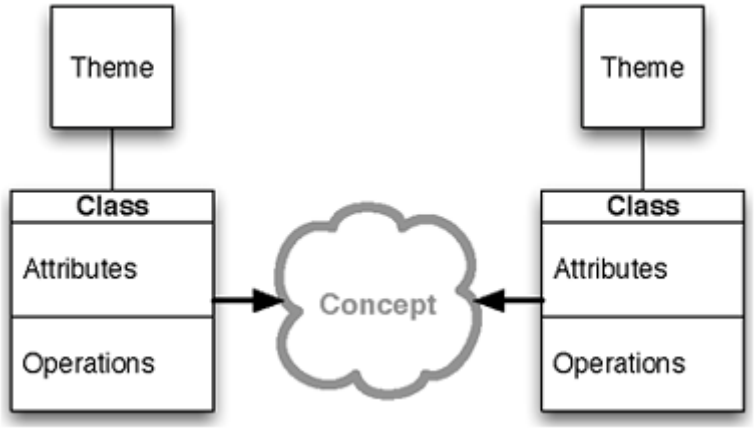

Figure 3-1 Themes related by concept sharing

In asymmetrical ***crosscutting*** the behavior in one theme is triggered by behavior in other themes, as seen in Figure 3-2. *Aspect* and *crosscutting* themes are used synonymously and are always themes that have behavior triggered together with behavior in other themes. *Base themes* are the themes that trigger aspect themes and might be themes that share concepts with other themes or even aspects themselves and have their own base.

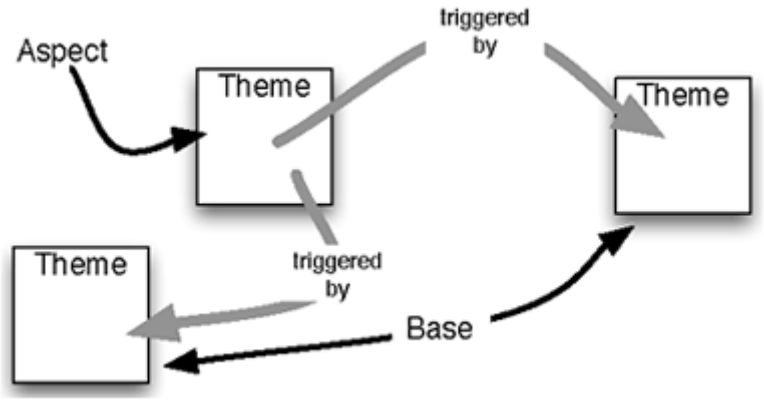

Figure 3-2 Themes related by crosscutting

While independent themes and themes that share concepts can operate without knowledge of one another, crosscutting themes require an abstract knowledge of the themes they crosscut and cannot operate independently.

The Theme approach [7], which lets one identify and design separate themes that are then combined forming the whole system, consists of two parts: ***Theme/Doc*** and ***Theme/UML*** as shown in Figure 3-3 and Figure 3-4.

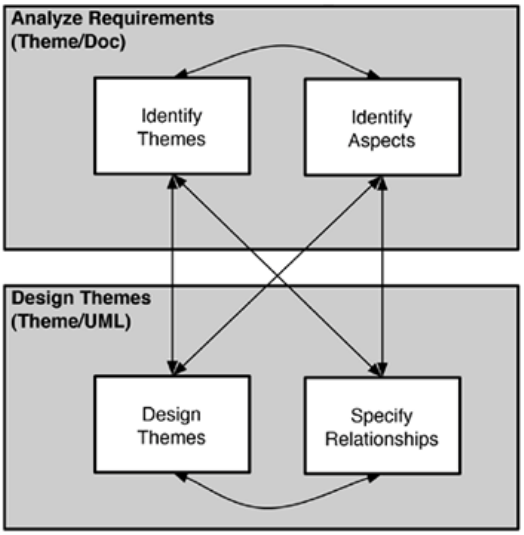

Figure 3-3 High-level view of the Theme Approach activities

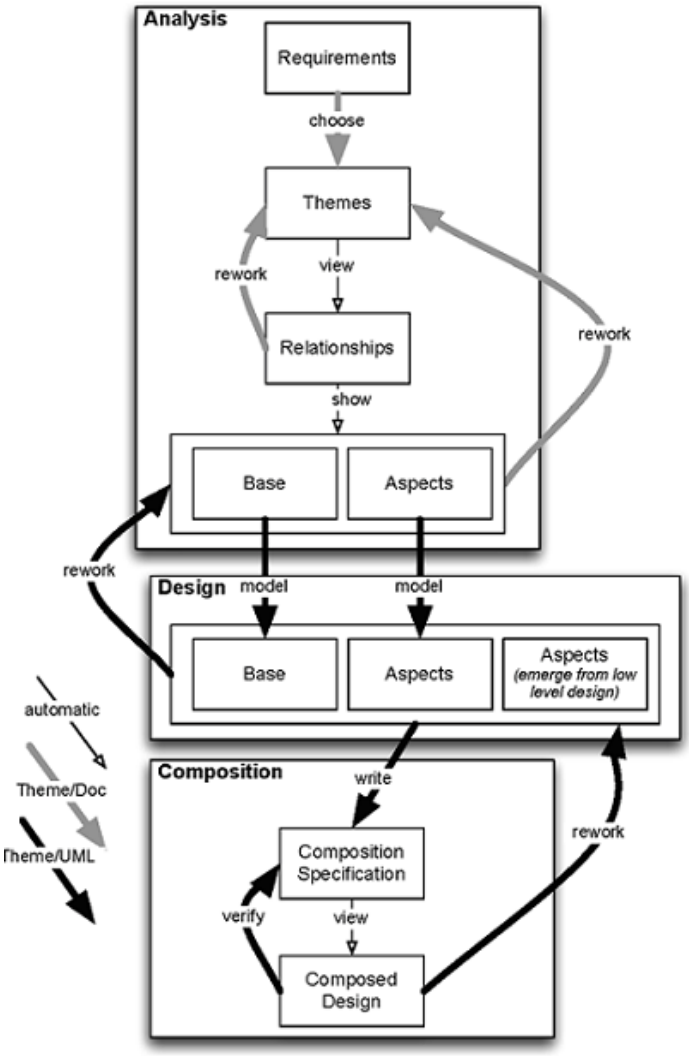

Figure 3-4 The Theme Proces

3.2 *Theme/Doc*

Theme/Doc assists in identifying the main themes in requirements documents and also provides heuristics for identifying, through the analysis of the themes' responsibilities, which of those are crosscutting or aspects.

Therefore, the process begins with an initial set of themes. The identification process involves sorting through a theme's responsibilities and trying to recognize a coherent set of behavior, as shown in Figure 3-5.

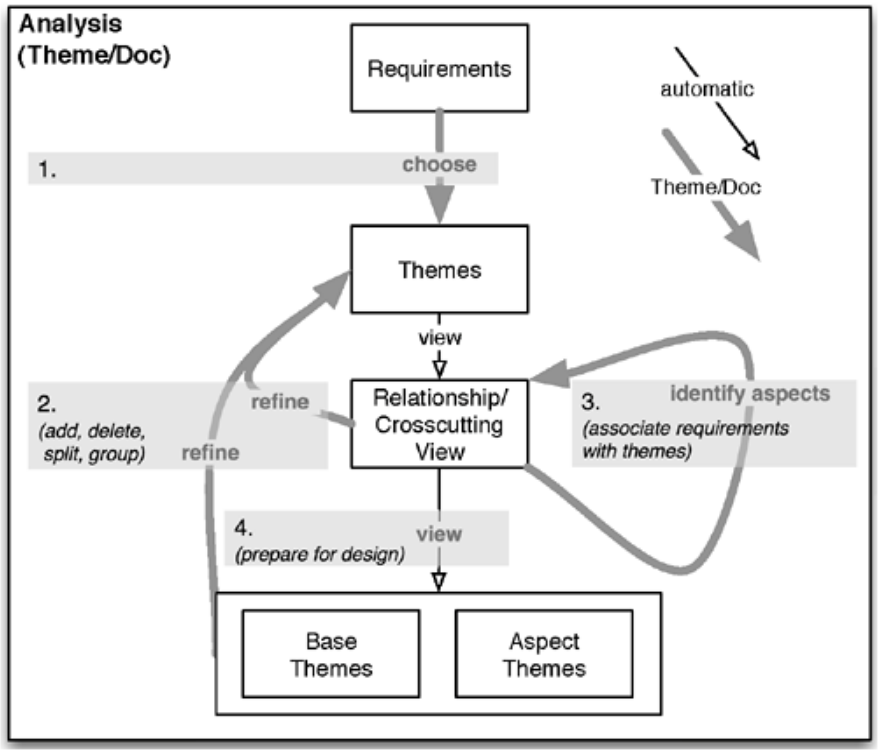

Figure 3-5 Theme/Doc process

To identify aspects using the Theme approach, one need to look for tangling in the requirements, that is, find themes that share a requirement. However, even if two concerns are described together in a requirement, this is not enough to identify as aspect. To locate an aspect we have to check several conditions:

1. The requirement has to be split up to isolate themes;
2. If one theme is dominant in the requirement, then the theme should be responsible for that requirement rather than the requirement being shared between themes;
3. If the behavior of the dominant theme is triggered by other themes mentioned in the requirement, then there is a trigger relationship between two themes;

4. If the dominant theme is triggered in multiple situations, then it is crosscutting and becomes the aspect; the triggering themes become the base.

3.3 *Theme/UML*

Each of the themes identified in the requirements merits separate design models through Theme/UML (see Figure 3-6).

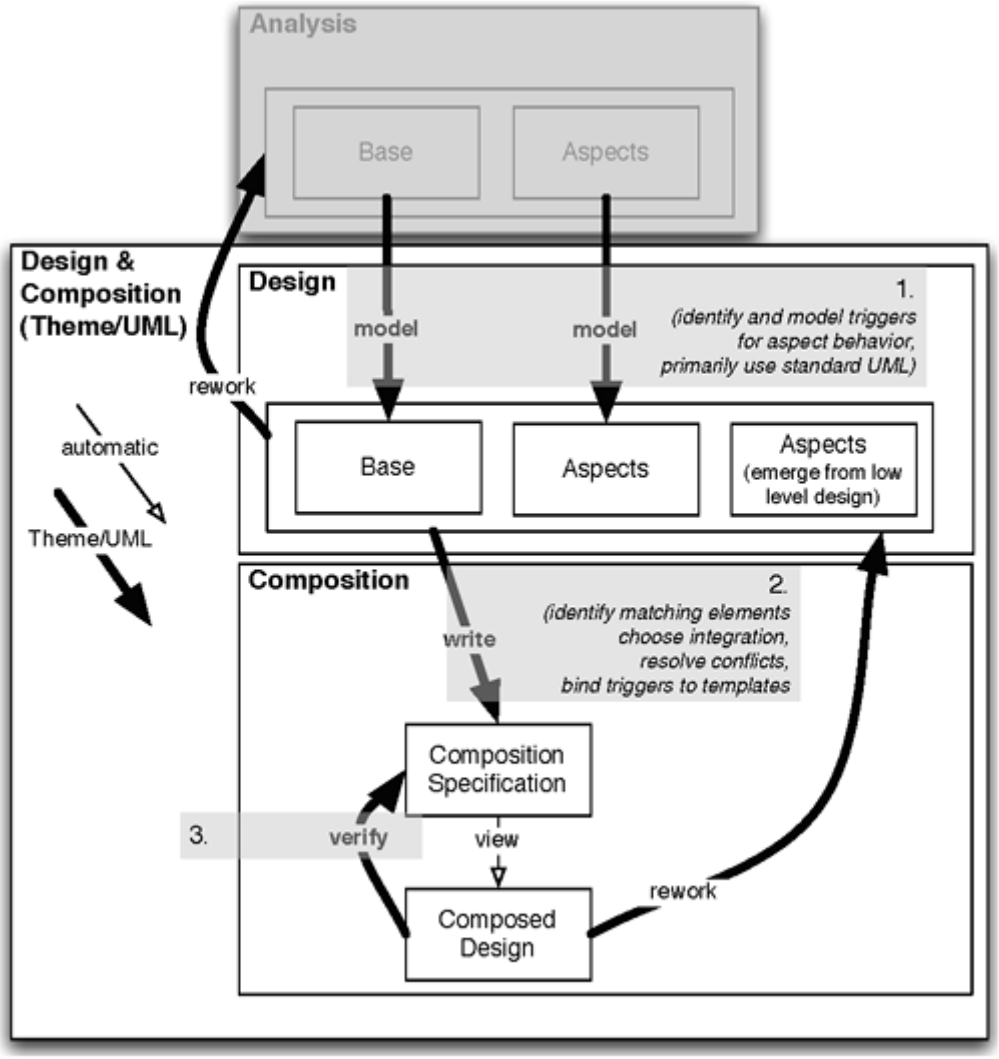

Figure 3-6 Overview of the Theme/UML process

In what concerns modularization, although Theme/Doc might identify some themes as crosscutting, or even if it detects some kind of overlapping between them, all of them can be designed separately. All the classes and methods pertinent to each of those concerns would be designed within the themes, making easier to work with a single feature or concern.

Modularized concerns require a way to connect to the rest of the system, so Theme/UML devised a "*composition relationship*", shown in Figure 3-7, that identifies those parts of the theme design that establish that connection. When crosscutting is involved, it is also required to identify when and where the dynamic behavior should happen and in the remaining types of overlapping, one must simply identify the correspondent theme designs and define their integration.

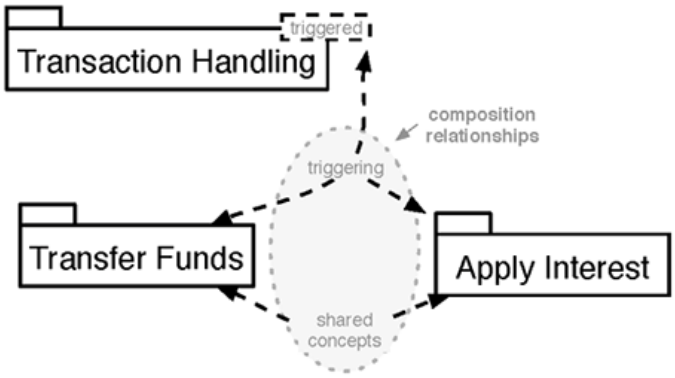

Figure 3-7 Theme/UML composition

The *individual-theme view* (Figure 3-8), showing all the requirements connected to one theme and also themes crosscut by it, helps to determine which objects and behavior should be modeled in Theme/UML.

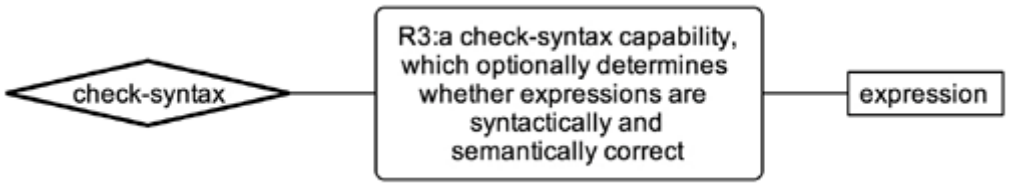

Figure 3-8 Individual Theme view

Theme/UML provides a means to consider the concerns of each theme separately by allowing one to work with separate design models for each concern. However, since all concerns must work together in an executing system, one needs to specify in some point the relationships between the separate models and verify those decisions. Thus, the goal is to have a modularized design, with the concerns separated wherever possible, constituted by a set of themes designs, each containing all and only those design elements that relate to the requirements the theme represents, therefore avoiding the phenomenon of scattering and tangling

In inter-theme relationships - *composition relationships* - one can identify two types of overlapping: aspect-oriented crosscutting identifies dynamic behavior that is similarly triggered across themes;

concept sharing relates to having design elements that represent a given core concept, scattered across themes.

When it comes to a single core concept being represented in the same way across themes, one can simply draw a composition relationship between themes, with a *match[name]* tag, as one can see in Figure 3-9.

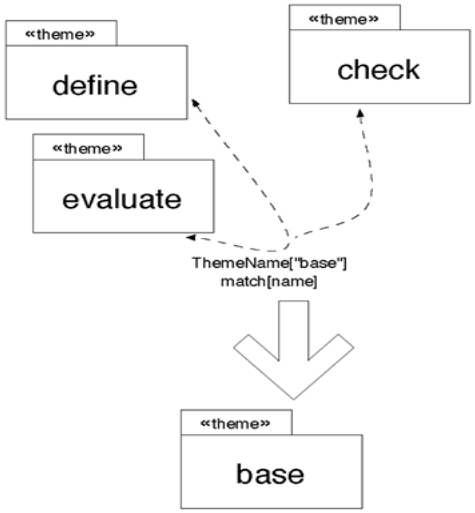

Figure 3-9 Composition relationship: concept sharing

On the other hand, when a theme contains a design for crosscutting behaviour there is a need for templates, which are used as placeholders for real design elements, that can be specified through the use of a Theme/UML - *tag bind[]*.

Having defined all the relationships between themes, a composition relationship can be applied to compose the design (see Figure 3-10). With the composed design, one can verify if the composition relationships are written as they were supposed to or even used it as a guide for a possible implementation in a non-aspect-oriented language. Despite of the fact that this approach intends to avoid non modular characteristics, a phenomenon known as tangling, the composed design will display them.

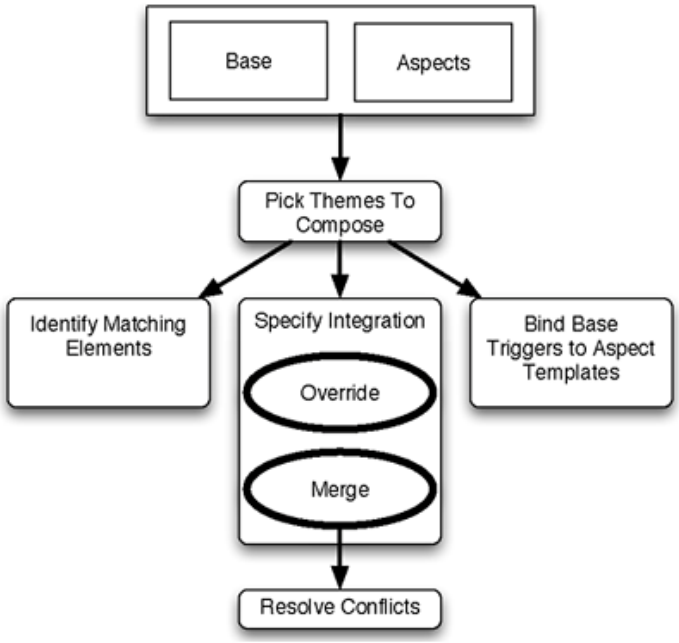

Figure 3-10 General view of the composition process

When composing themes, design elements that have the same name are defined as relating to the same concept, being merged on the output. All the classes, attributes and methods belonging to the input classes that match will appear in the composed output. In what concerns relationships defined in the input themes, they are all added to the output; however, all redundant relationships are excluded and the duplications are added just once. This duplication does not include aggregation relationships that appear for a subclass and also for its super class, since they are not classified as duplicates, and are thus added. To specify triggering operations, there are template parameters, signaled by "< >", as shown in Figure 3-11, which are applied to aspect themes. For every triggering operation a sequence diagram is drawn according to the specification present in the composed theme.

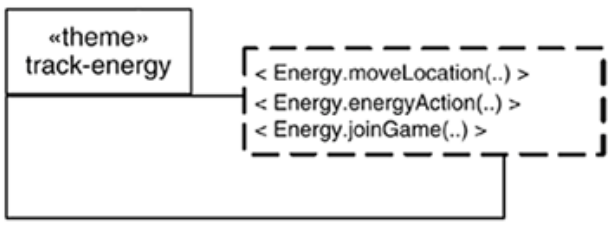

Figure 3-11 Template notation

3.4 *The Theme Process*

The Theme approach treats concerns as a unit of modularity in a way similar to the object-oriented treatment of entities; each of those concerns becomes a theme, be it a base or a crosscutting one.

In order to identify the previously mentioned concerns, one needs to elect a set of them from the system's requirements and iterate over that same set, adding, deleting, splitting or even grouping themes that come up, evaluating their potential to be treated and modeled separately (Figure 3-12and Figure 3-13).

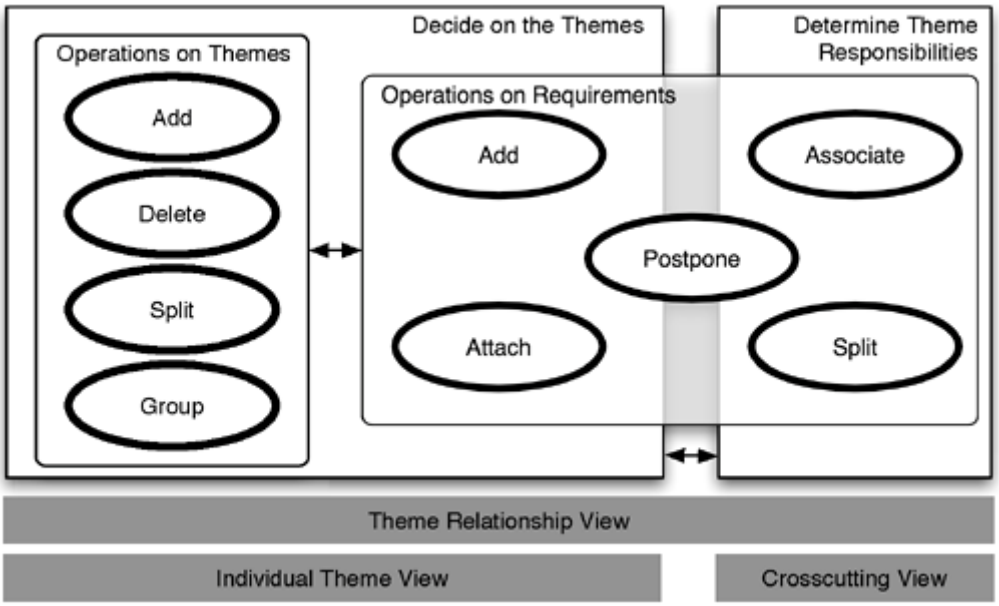

Figure 3-12 Activities involved in choosing themes and deciding on their responsibilities

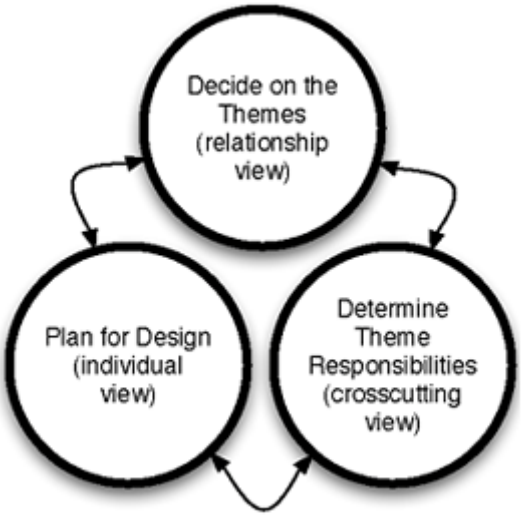

Figure 3-13 General view of the analysis process

3.4.1 Deciding on Themes

The starting point for the Theme/Doc approach is finding a set of potential themes of your system, that is, identify all the behavior described in the requirements that could potentially be a theme. There are several approaches to apply on this stage:

1. ***Choose Carefully Up Front Approach:*** this approach consists of reading through the listed requirements and pinpointing key descriptions of functionalities that may become system features; it is therefore a tiring approach at its beginning, seeing as one must assess with special care what concerns are described by the requirements. The main task becomes splitting up the themes that are more abstract or whose requirements do not form a cohesive set of functionality when grouped.

2. ***Start With Everything Approach:*** this approach consists of choosing all verbs, or actions words, written in the requirements, saving time during the first step of going through the documentation, however resulting in a possibly huge set of potential themes. The main task here is to identify which of the chosen terms are real themes and to group verbs, thus forming bigger themes.

3. ***Combination of the two approaches:*** like the name says, this approach is neither extremely conservative nor extremely liberal and consists of reading through the list of requirements and identifying behavioral patterns, concerns or features in their possible representations; the set of themes obtained should only require some grouping and segmenting in order to become a good set of themes.

3.3.2 Operating on Themes

On the initial set of themes, it is possible to find behaviors associated with a particular theme that do not belong together, which is not desired since the themes must be coherent and cohesive.

Applying the Start with everything approach will give rise to many finely grained themes - some of the names of these themes may have the same literal meaning or might be synonymous in the sense that two events always happen together - so the solution is to regroup them into a larger theme. On

the other hand, it is possible to find some themes too trivial or too unrepresentative of the system's functionality to keep - those themes should be deleted.

3.3.3 Operating on Requirements

When a requirement is not that interesting from a design perspective, it can be postponed until more information becomes available.
Requirement adding might be done in a normal way, simply introducing them into a given set, or they might be the outcome of the splitting of existing ones. Either way, new requirements may influence a set in a way that generates new themes, edits the ones already existent or even change their grouping.

3.3.4 Themes, Requirements and Aspects

Using this approach, one must achieve multiple themes, each of them being responsible for implementing a cohesive set of system requirements. Despite of the possibility for themes to implement many requirements, those should only be implemented by only one theme.
After ensuring that a shared requirement cannot be split to untangle multiple themes, there are three rules to apply to see if the requirement reveals an aspect:

1. If one of the themes dominates a shared requirement, then the dominant theme may be an aspect;

2. Aspect behavior is triggered by some behavior in the base and can happen implicitly at triggers locations; if there were no aspect/base relationship, then the behavior would have to be triggered explicitly from the location in the base;

3. If a theme's behavior is triggered in only one place, then it doesn't make sense to make it into an aspect.

Just because a theme crosscuts another theme, it does not mean that the crosscutting theme will always crosscut others or that the base theme chosen will always be the base theme – a theme can be crosscut and crosscutting at the same time.

For theme refinement there is a Theme/Doc view called theme relationship view or relationship view, as shown in Figure 3-14.

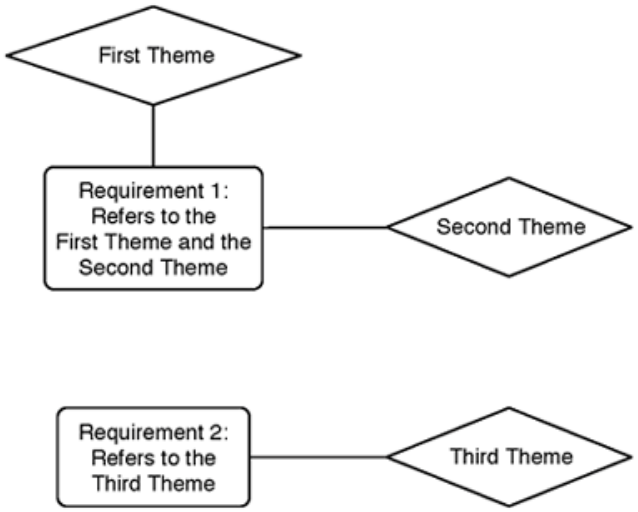

Figure 3-14 Abstract example of a Theme/Doc relationship view

In order to better understand what a theme is and how themes and requirements should be related let's consider a brief example [7], involving the construction and evaluation of a simple expression evaluation system (EES). The following requirements from [7] must be considered:

- R1) An evaluation capability that determines the result of evaluating an expression.
- R2) A display capability that depicts expressions textually.
- R3) A check-syntax capability that optionally determines whether expressions are syntactically and semantically correct.
- R4) The check-syntax, display, and evaluation operations should all be logged.

Analyzing the requirements, one could identify a set of action words: "evaluation", "display", "check-syntax" and "log" – each of these defines a potential theme.

Each theme was identified via some requirement, thus they must be linked.
Regarding the "evaluation" theme, it was identified in requirement R1; "display" was identified in requirement R2; requirement R3 identified "check-syntax"; finally, requirement R4, which identifies the "log" theme, also refers other themes in its description, namely "check-syntax", "display" and "evaluation". All these relationships can be represented through a link, connecting the theme with the requirement (or requirements) that mention it.

Regarding the presented requirements and identified themes, the resulting relationship view can be seen in Figure 3-15.

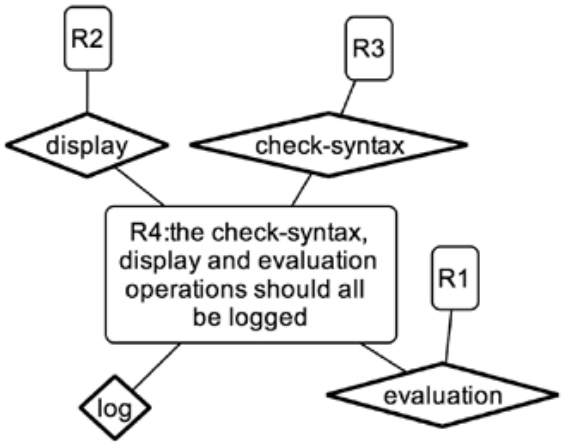

Figure 3-15 Example - relationship view

Aspects can be identified by looking up requirements that are shared between themes. Crosscutting relationships are shown in a special Theme/Doc view called a *crosscutting relationship view*, as shown in Figure 3-16.

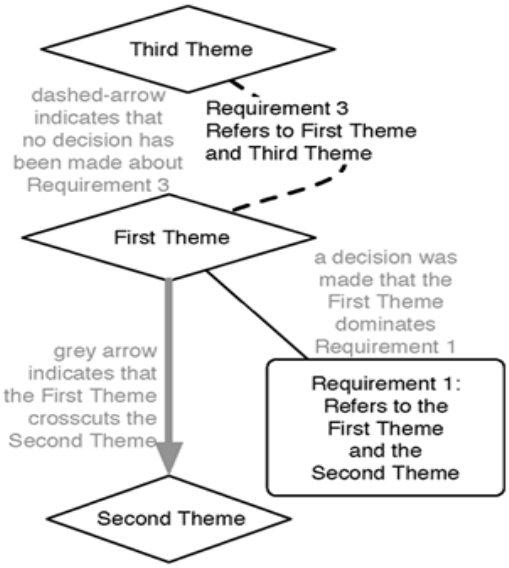

Figure 3-16 Abstract example of a Theme/Doc relationship view

Again, considering the EES system, we can apply the defined heuristics to identify crosscutting behavior.

First, Figure 3-15 shows that R4 is a shared requirement, then some of the linked themes should be responsible for it. Analyzing the R4 description, we can see that the “log” action is a necessary action when considering “check-syntax”, “display” or “evaluation”, thus being the responsible

theme for R4. It also defines a crosscutting relationship with the other mentioned themes, affecting its behavior. Theme represents a crosscutting relationship with a grey arrow, from the theme that affects other theme behavior to the affected theme. To show that a theme is responsible for a requirement, a simple link is used.

Figure 3-17 depicts a crosscutting relationship.

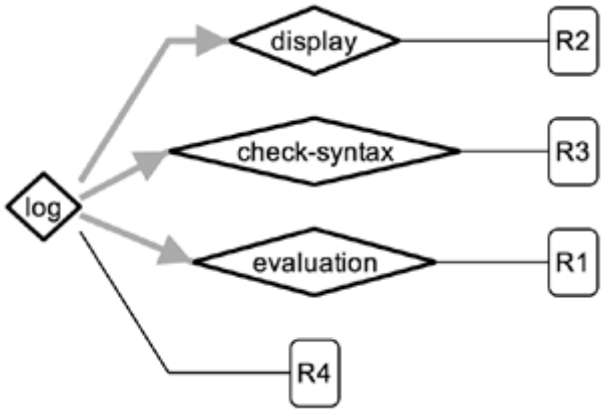

Figure 3-17 Example - relationship view with crosscutting

To compose the "log" behavior with the other themes (base themes), one should specify where, in the base, the crosscutting behavior should occur. Thus, in the "log" theme, a template should be placed, specifying which "parameter" should be replaced in the design by the elements in the theme to be crosscut; this parameter is the one responsible for triggering the crosscutting behavior. To specify that replacement, a "bind[]" tag is used, specifying the methods to be considered on the composition: these methods identify pointcuts, i.e., points in the execution where behavior can be join.

For instance, Figure 3-18 shows that, when evaluating the method "Expression.check()", the "LoggedClass.loggedOp()" is triggered, i.e., logging will occur when expressions are checked.

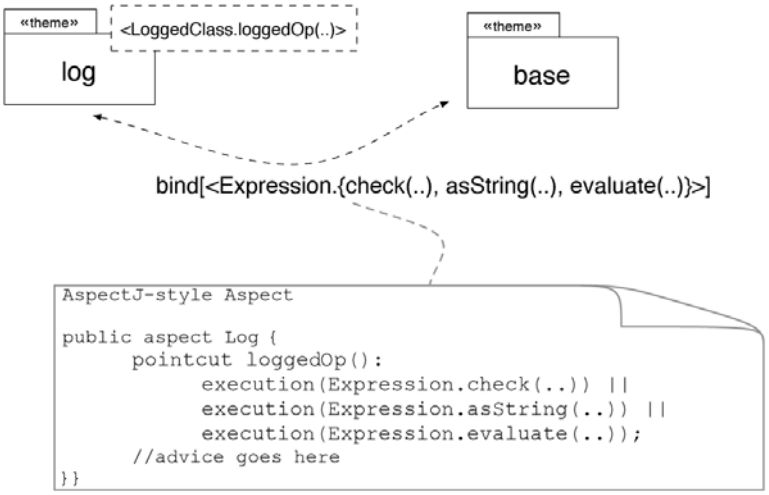

Figure 3-18 Example - Composition relationship

3.4 Adding new functionalities

Contrary to what happens in the object-oriented approach, a new functionality can be added to a system without making any changes to the existing design, since each concern, as derived from the new requirements, is considered to be a theme in its own right. So, new themes can be designed separately and composed with the full design using a composition relationship, as shown in Figure 3-19.

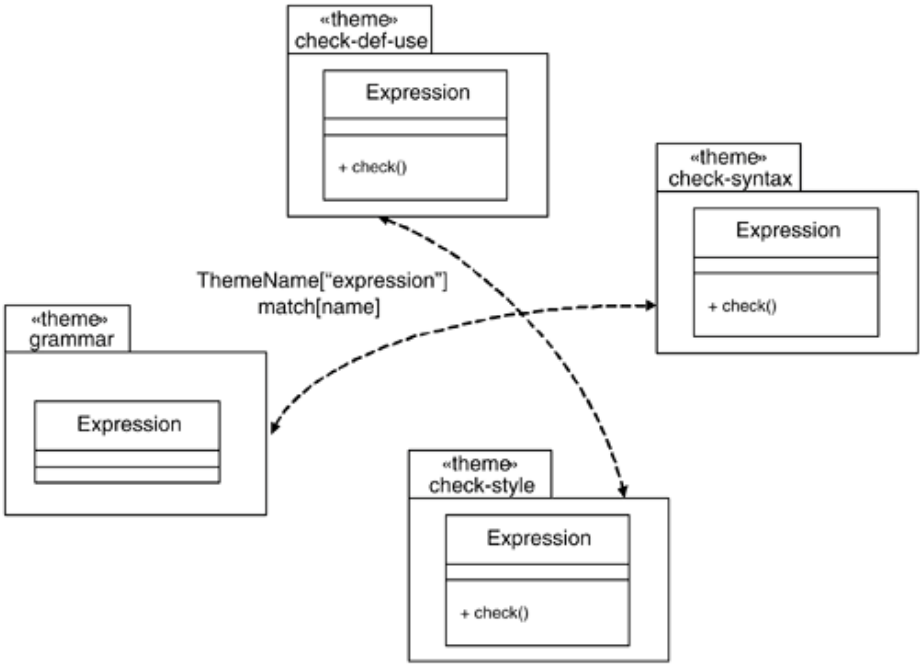

Figure 3-19 Composing new themes

The whole approach is designed to ensure that separate themes are obtained from their own perspective, without reference to other themes. Therefore, whether a theme design was created during the development of an initial version of the system or for later versions makes no difference to the composition process, being the themes concept-sharing or crosscutting ones.

3.5 Summary

This chapter has described the Theme Approach and its main processes: Theme/Doc and Theme/UML. The whole process was also explained, more precisely the methods to select themes, to operate them and on its requirements, and the identification of aspects.
Since the purpose of this dissertation is an integration between the Theme approach and aspectual scenarios, the next Chapter will introduce the Scenarios issue.

Chapter 4
Scenarios in System Development

Scenarios provide developers with a powerful tool to deal with the growing complexity of systems and their analysis. They help ensure everyone with an interest in the project, known as stakeholders, that they will be provided with an ample view of all important aspects of the solution. Scenarios might be seen as stories, either short and succinct or complete and descriptive, laying out the idea behind the sequence of actions an agent will carry out and are applicable to all different kinds of system configurations, at any stage of their development.
Since scenarios are intimately related to the people involved with the project, one needs to explicitly define the concept of stakeholders. They are, as said previously, a group of people interested in the project, but more than that, they have a stake in the project's outcome, and so this interest becomes a legitimate one. Also, they may represent diverse groups, with diverse notions and concerns related to the project. Therefore, scenarios provide the development team with a way to communicate with the stakeholders, guaranteeing that everyone's demands are being met as well as possible.
Scenarios also involve people in the requirements discovery process, by proposing a scene for them to explore and elicit them naturally.

Regarding the thematic of the dissertation, extra emphasis will be given to the approaches related to aspectual scenarios and misuse cases.

4.1 *Main application of scenarios*

All of the following approaches can be found in [1].

4.1.1 Scenarios in requirements discovery

This approach uses business events, business use cases, product use cases and scenarios to discover and organize requirements. Furthermore, a requirements knowledge model is constructed, providing a language to group and link requirements.

A scenario describes what is supposed to happen in a specific system. A business event is something that happens and that has a pre-planned business response, which in turn, is a business use case. A business use case, shown in Figure 4-1, contains all the related business requirements.

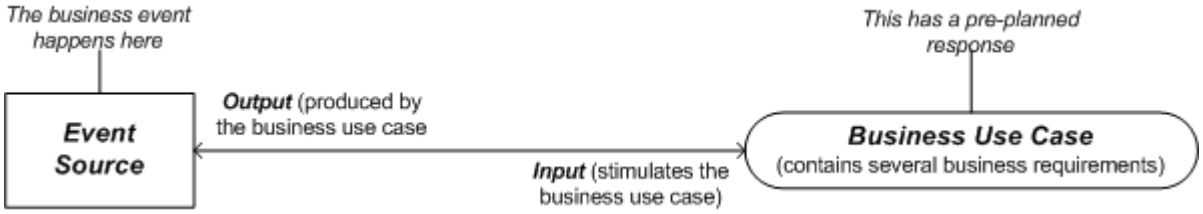

Figure 4-1 Business use case

Requirements can be discovered by building scenarios for the business event response, i.e., the business use case. *Normal Case, Alternative Case, Exception Case* and *What-Ifs* are the main types of scenarios. Normally, the process starts with the normal case. In order to do that, the following points must be considered, not necessarily in this order:

- Identify a business event;
- Identify the business use case;
- Identify the stakeholders;
- Write an informal description of the business use case;
- Identify scenario steps based on the previous description.

A ***business event*** has a name, which helps to understand what the business needs to respond to; a number, used to trace the event and connect it to the business use cases and atomic requirements; *input data*, caused by the business event; and *output data*, which is produced in response to the business event.

Since the business event causes a business use case, when a business event is identified, so is the corresponding business use case. A *business use case* has the same attributes as a business event, however the *input* and *output* data in this case refer to which of the business use case must respond and to what is produced by the business use case, respectively.

Having defined the business events and use cases, it is time to identify who is responsible to provide the input for the scenario - the ***stakeholders***. To each business event a business owner is associated, that will help to find all the requirements for that business use case. In this step, one must identify the business event and corresponding use case, the owner and other stakeholders possibly involved.

In possession of all this information, a scenario can be built, and then partitioned into a number of steps, thus reflecting the current understanding of the business use case. Each of these steps can be questioned in order to find alternatives to the normal case scenario ("*Does this step always happen precisely as stated?*"), exception cases ("*What data conditions could make the step unable to proceed?*") and/or what-ifs scenarios ("*What would happen if this constraint did not exist?*").

Scenarios are useful for discovering the atomic business requirements, besides providing logical business grouping for individual requirements, which connects to a product grouping, leading to the resulting system requirements.

A requirement has several attributes, as depicted in Figure 4-2 [1].

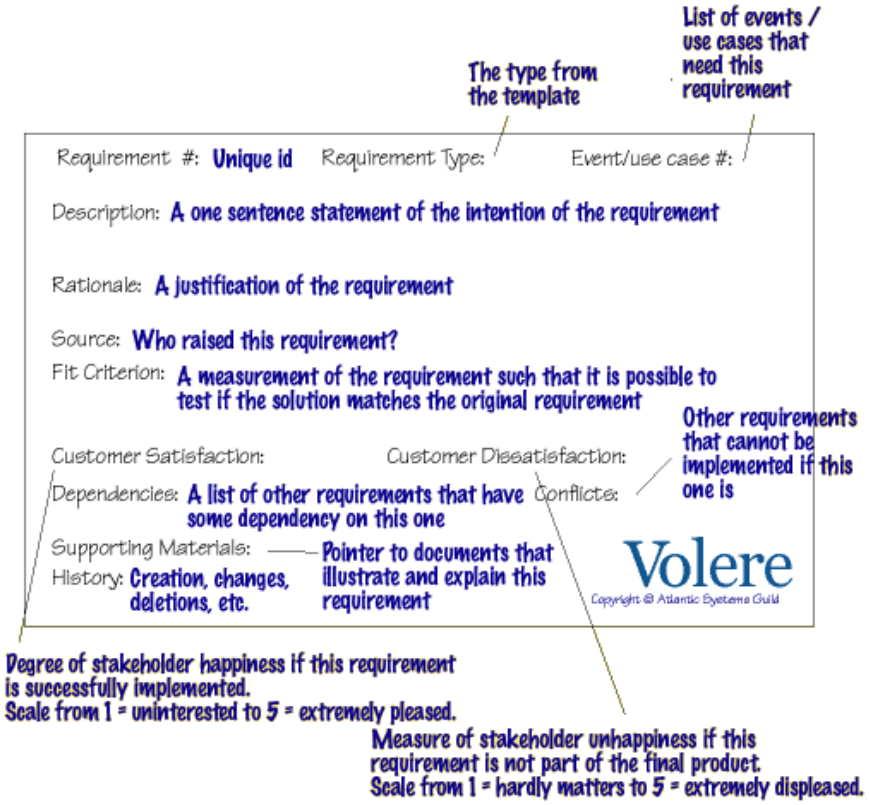

Figure 4-2 Atomic Requirement in the Volere Template

Finally, a Requirements knowledge model is shown in Figure 4-3 that summarizes connections between atomic requirements and other requirements-related knowledge.

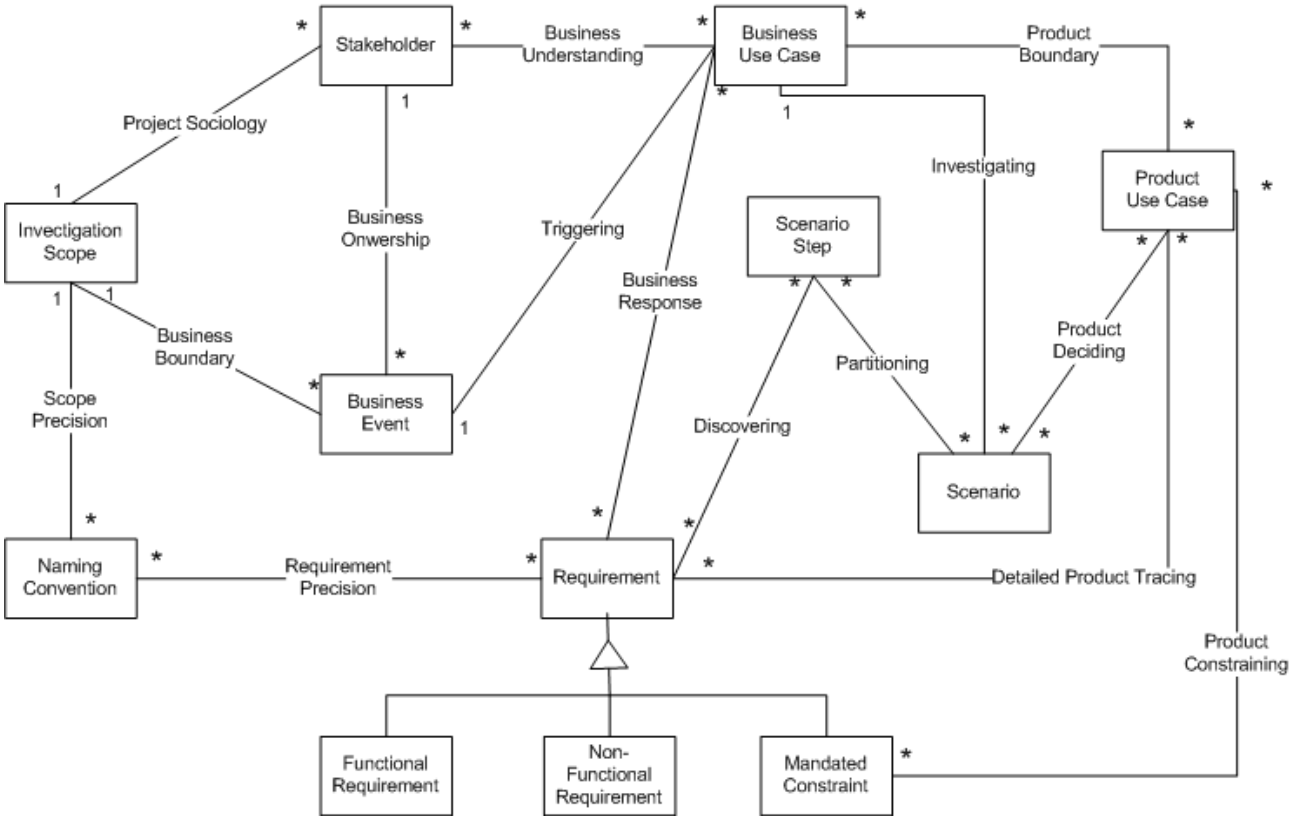

Figure 4-3 Requirements Knowledge

4.1.2 Authoring Use Cases

A Use Case can be seen as a goal that the user shares with the system, thus its name must be the goal's name. When associated with Use Cases and scenarios, goals identify their content, define their scope and clearly specify what the system should do, thus constituting their most important attribute.

The scope of a scenario or a Use Case can be defined by delineating the main goal that the scenario or use case describe, identifying the different ways to achieve that goal and associating a scenario with each of those ways.

Associated with a Use Case can also be preconditions, dependency links, system requirements, priorities, sequence diagrams.

The most important aspects in Use Cases concern their general structure, their contents and the style chosen to present them. This approach proposes three series of guidelines to assist Use Case authors:

- **General Use Case guidelines:** these guidelines include all of the most used approaches, like "describe each scenario separately", "identify Use Cases by means of goals", etc.

- **Scenario contents guidelines:** indicate the "what" of the scenarios in Use Cases, advising the author about the expected content.
- **Scenario style guidelines**: indicate the "how" of the scenarios, aiming to help authors to adopt a good style.

Content and style should not be seen as two independent concepts, but as complementary ones: style guidelines should not be applied without taking care about content, as content guidelines cannot improve scenarios without correct style.

4.1.3 A Scenario-based design method for human-centred interaction design

This approach uses four different types of scenarios: user stories, conceptual scenarios, concrete scenarios and use cases. The goal is to develop a usable, useful and engaging interactive system.
The method is shown in Figure 4-4.

User stories are "real world experiences", being useful to understand what people do now and want to do in the future as well as what are their problems.
Conceptual scenarios, being generated by analysts and designers and with the participation from users, are useful for generating ideas.
Concrete scenarios are also generated by analysts and designers by introducing design constraints and specific interaction methods to the abstract scenarios, being useful to evaluate alternative physical and logical designs.
Use cases, constituted by several concrete scenarios into more generic scenes, are generated by the analyst/designer as part of the formal specification of the designed system.

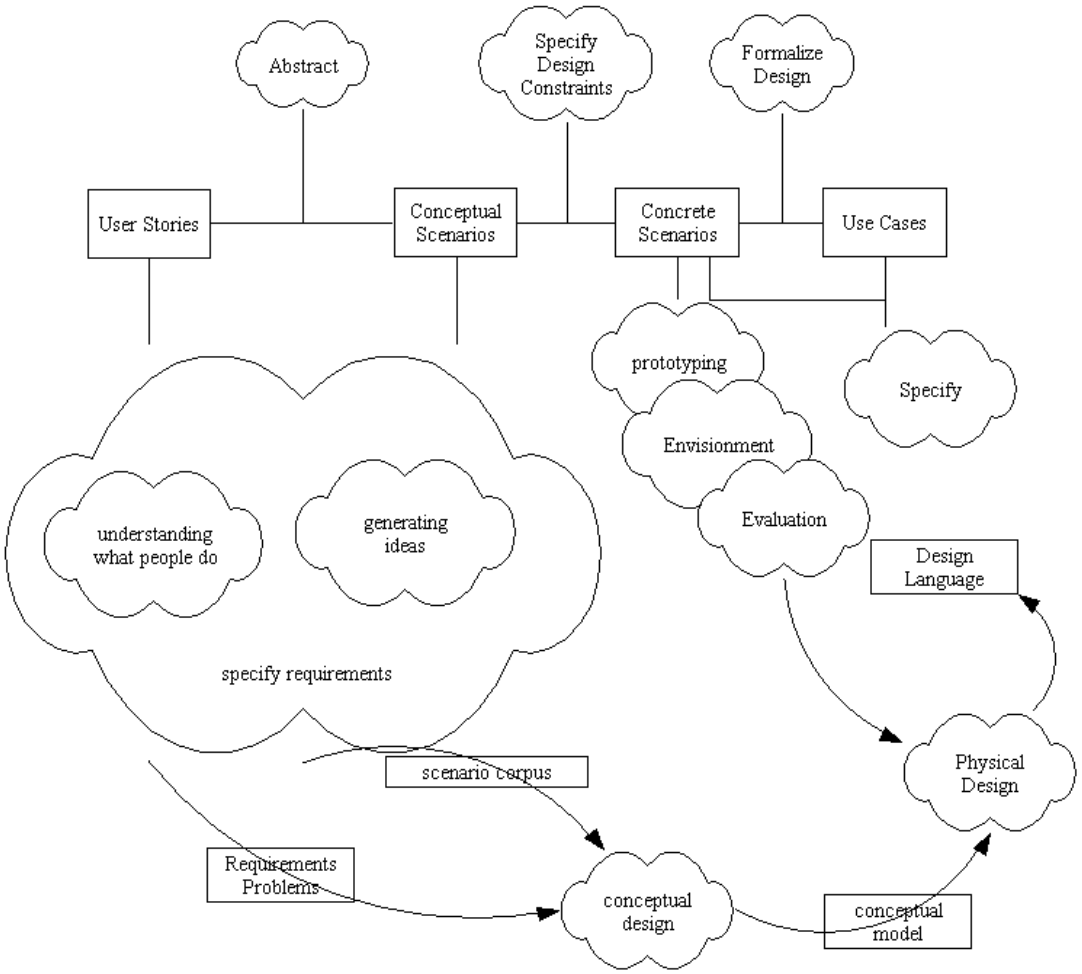

Figure 4-4 Scenarios throughout the life-cycle

The main activities that compose the method include:

- *requirements elicitation* via interviews, ethnographic methods, etc;
- *abstraction*, thus grouping stories into meaningful conceptual scenarios and defining a language to be shared by designers, developers and users;
- *interaction design*, which defines how the logical processes and data are distributed between devices and people;
- *establishing design constraints*, which will result in the "look and feel" of the system;
- *object/action analysis*, establishing a conceptual object model from the scenario corpus.

4.1.4 User stories in agile software development

In this approach, a set of user stories is written on cards, to which are given a name and a number. Each story has an estimate time to be realized in order to be release. A story can be split into smaller stories, if the associate estimate time is considered to be too long, or if one aspect of the

story has a higher priority. All the "resulting" stories are then prioritized and aggregated into collections, which will be associated to a team in a certain period of time. The products are then released and analyzed concerning their validity, the developed system is used by the customer and possible new stories are considered.
These stages are repeated until all the stories have been analyzed, specified and developed.

4.1.5 Use Cases, Test Cases

When a Use Case model is constructed, all the possibilities are considered, whether they are alternatives, exceptions or the normal case, covering all of the expected (or not) behaviors. Since each use case describes a complete set of actions in order to achieve a certain goal, it makes sense to check if this referred goal is effectively and correctly achieved. The solution is to perform a test case that will validate the current specification. However, in order to cover all the variant courses, the Use Case must be well constructed. Otherwise the current approach will not work. If it is not the case, the Use Case must be improved or another basis for testing must be found.
Thus, given a complete Use Case model, a test case is generated for each possible branch, considering all the alternatives and/or exceptions of the normal case.
This approach constitutes a useful technique to requirements validation, integration and testing.

4.2 *Negative Scenarios and Misuse Cases*

A Misuse Case [1] constitutes the description of a negative scenario in a representation similar to that of a Use Case in a way that it is perpetrated by an actor and has an effect on the system. In a Misuse Case the actor is a hostile agent that is attacking the system under design and is, almost always, of human origin. In Use/Misuse Case diagrams, one can represent the relations between these two forms of representations, constituting an important form of documenting threats and possible consequences to the system, thus providing the developer with an important security analysis and a way to obtain new subsystems, granting Misuse Cases with a role in the design process.
Through Misuse Cases, shown in Figure 4-5, one can also obtain requirements for a system, even more so when exception cases might be overlooked. Therefore, the scenario approach to negative situations that might occur in a system is a useful tool to identify security and safety related

requirements, system failure situations and cases to be tested, providing a uniquely suited means for probing the unknown and unexpected.

Figure 4-5 Misuse Case

However, although the fact that it combines an unrestrained and adaptable user end with a well structured system end model makes it a useful approach, the fact that it is essentially based on a human point of view and thus qualitative in its nature, makes it difficult to guarantee that all system threats are discovered. Also, it does not offer a way to combine the several threats that might come up, focusing its attention on one threat at a time.

4.2.1 Misuse Case Analysis

This particular analysis consists of a three step plan that begins with a first-cut Use Case model, consisting of the basic system goals and a graphic representation of the normal course of action for the system. The second step corresponds to the identification of hostile agents that might target the system and the last step is the consequent elicitation of the Misuse Cases that such agents might originate. The last two steps can be seen as an important aid to finding Exceptions and the scenarios required to handle them.

4.2.2 Eliciting hostile roles

As mentioned in the previous section, the identification of hostile roles, agents that incur in actions that might bring harm to the system, its stakeholders or their resources, is an important step in Misuse Cases modeling; their representation is similar to that of normal Use Case actors, with a role name and a brief description of their viewpoint.

4.2.3 Iliciting misuse cases

Misuse Cases representation is similar to that of Use Cases, consisting simply of the names of their goals, the difference being that these are hostile in nature.

When including Misuse Case modeling in a Use Case diagram, it is important to indicate which of the existent Use Cases are affected by the intruding hostile behaviors, thus new types of relationships come up whose names describe the effect they have on the target Use Case, these are: ***threatens***, from a Misuse Case to a Use Case as shown in Figure 4-6, and ***mitigates*** in the opposite way. On the other hand, the positive oriented Use Cases might be able to interfere with the occurrence of certain Misuse Cases, thus the ***prevents*** relationship, or simply monitor that a Misuse Case occurs, originating the ***detects*** relationship.

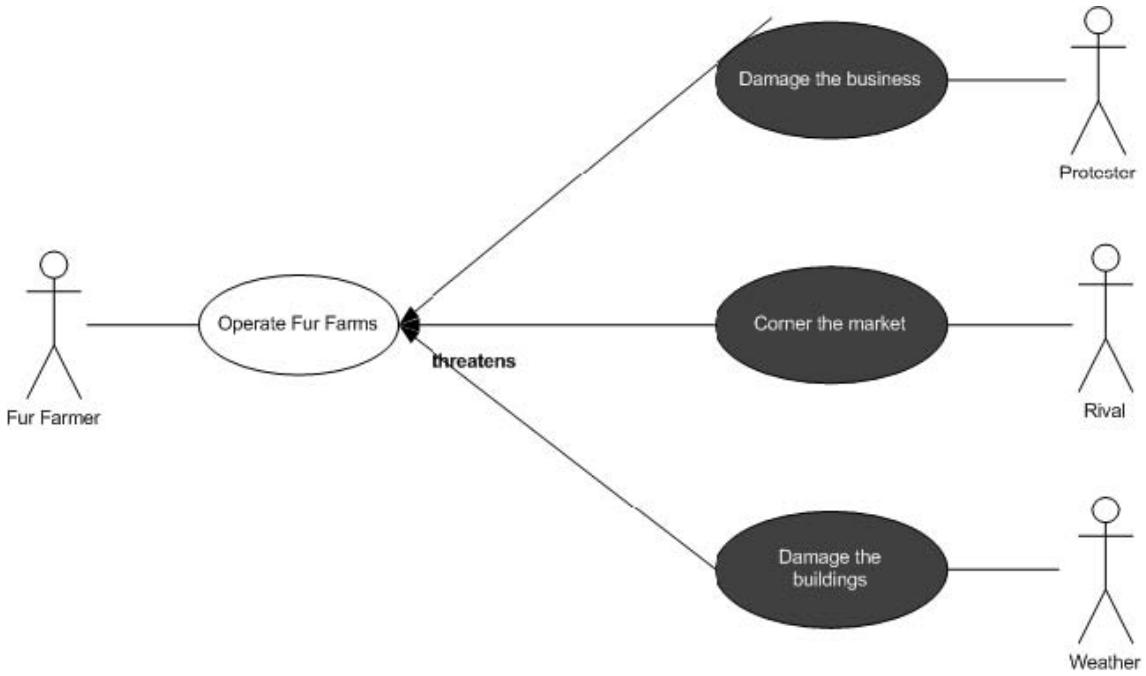

Figure 4-6 Documenting threats with use and misuse cases

4.2.4 Eliciting exception scenarios, requirements and further use cases

When modeling Misuse Cases, one must first approach the requirements document to check if the Exception Events that certain hostile agents might perpetrate can be prevented by any of the existing Use Cases.

When these Exceptions are compiled, one must decide on how to deal with them and, for this, two options exist:

1. One can automatically add exception-preventing or handling requirements to the Use Case affected by the Exception Event, adding to the overall qualities of the system when it comes

to security or safety and thus justifying the requirement insertion with the Misuse Case interference;

2. Situations that are more complex (see Figure 4-7) in nature require a more complex approach to handling the Exception events; one must gradually analyze all Use Cases and identify those that are more likely to be able to mitigate the threats that the Misuse Cases pose. These actions that the selected Use Cases might be responsible for, will be described in their primary scenarios and the **mitigates** relationships can be drawn to the Misuse Cases that are being monitored. To complete this, the **has exception** relationships can be drawn from the Use Cases threatened by the Exception Events to the threat-mitigating Use Cases.

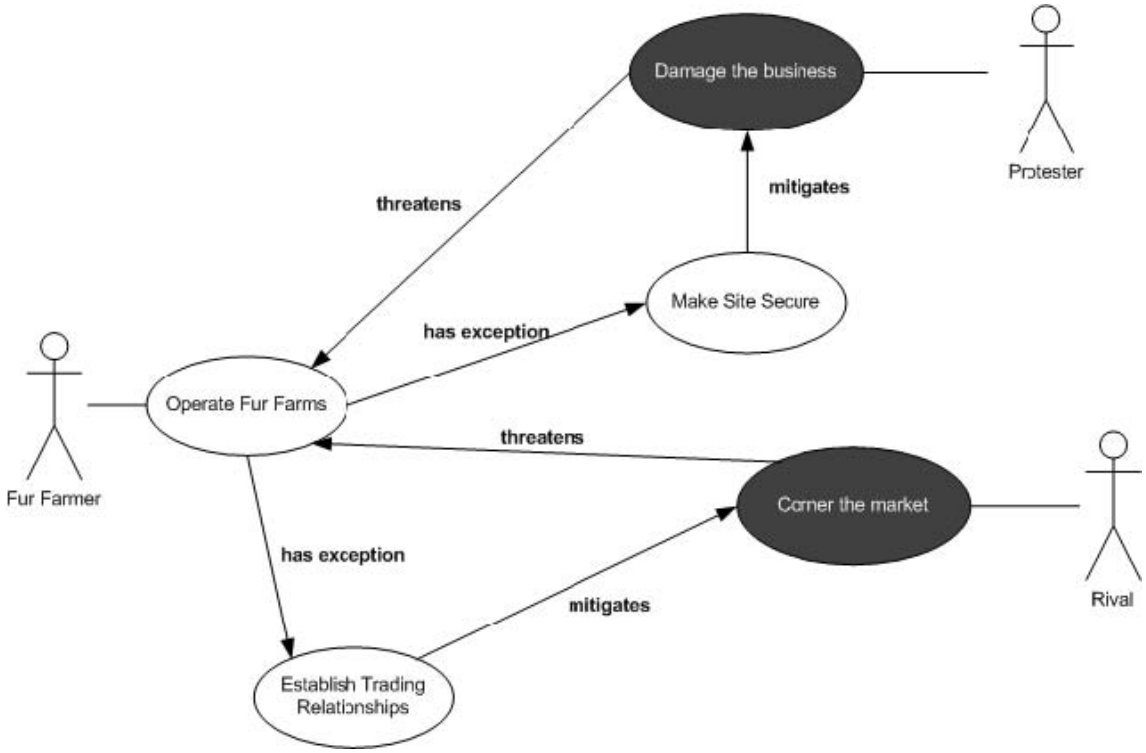

Figure 4-7 Documenting exceptions

4.2.5 Use/Misuse Cases relationships

When one integrates Misuse Cases in a Use Case diagram, four "basic" combinations might come up relating both types to each other or to themselves, as seen in Figure 4-8.

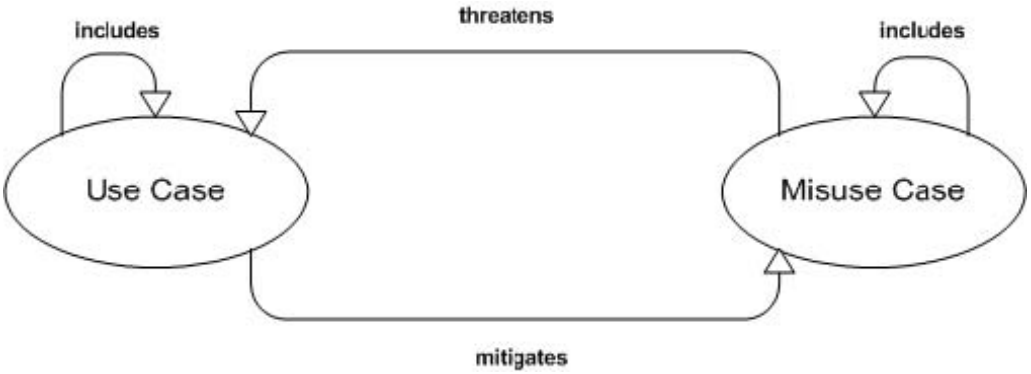

Figure 4-8 Use/Misuse Cases relationships

4.2.6 Design trade-off and conflict analysis

The Misuse Case approach can also be used in trade-off and conflict analysis during the design stage. Conflicts in this stage are mainly the product of not knowing the best way to meet certain requirements, even more so when there are incompatibilities between design elements (shown as **conflicts with** relationships), with sometimes negative effects shown as **aggravates** relationships) on problems already present in Misuse Cases.

4.3 *Aspectual Scenario-based approaches*

Scenarios can also be used to specify crosscutting behavior, thus modeling the systems in terms of their aspects. There are several approaches that consider aspectual-scenarios as a way to model the system under consideration: these approaches are based in patterns specification, graph transformations and use case maps.

Again, considering the focus of this dissertation, more emphasis will be given to the first two approaches: "*Scenario modeling with aspects*" and "*Modeling Aspects using a Transformation Approach (MATA)*".

4.3.1 Scenario modelling with aspects

The approach described in [25] aims at defining a way to compose scenario-based requirements and aspects, also represented as scenarios. To represent aspectual scenarios an ***Interaction Pattern Specification*** (IPS) will be used, whereas UML sequence diagrams will represent non aspectual scenarios. The presented technique will compose aspectual and non aspectual scenarios using an

instantiation of the IPSs and also a state machine synthesis algorithm, which will result in a set of state machines that represent the composed behavior from the aspectual and non aspectual scenarios.

Introduced in [10], pattern specifications (PSs), based on the *Unified Modeling Language* (UML), can be seen as a way of formalizing the structural and behavioral features of a pattern. Each element in the PS is a role, which specifies a subset of the instances of the UML metaclass. An IPS consists of *lifeline* and *message roles*, defining a pattern of interaction between its participants. Role names are preceded by a vertical bar in order to be identified as roles. A conforming sequence diagram must instantiate each of the roles with UML model elements, also taking in account multiplicity issues and other constraints. However, any number of additional model elements may be present so long as the role constraints are maintained. A sequence diagram is an instance of an IPS if the relative ordering of the messages and their participants are preserved. As an example, there is Figure 4-9.

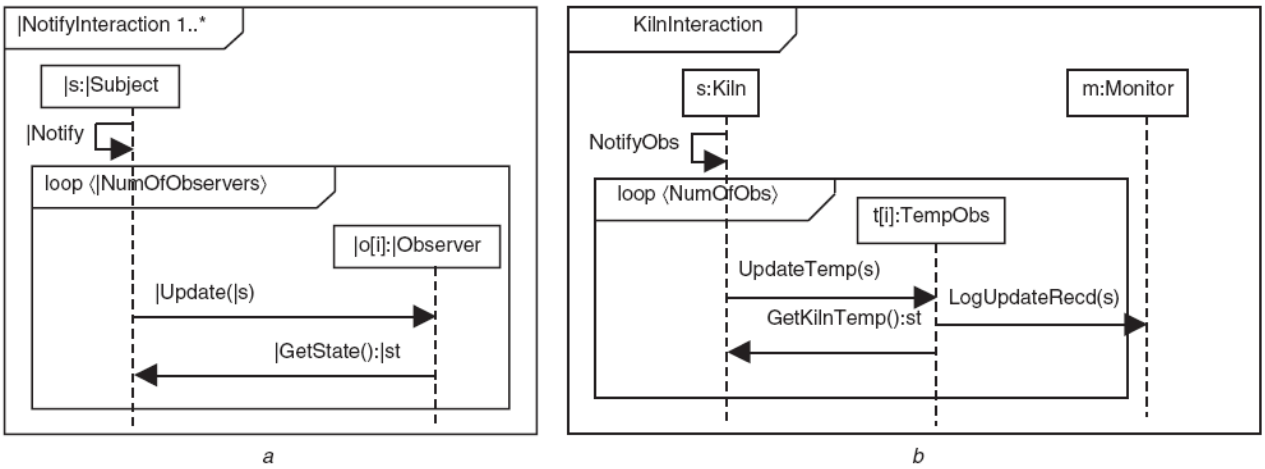

Figure 4-9 IPS (a) and a conforming sequence diagram (b)

In what concerns the approach, the process starts with a possibly incomplete set of requirements.

From these requirements functional and nonfunctional concerns will be identified, the first ones described by use cases and the last ones specified in templates, like the one shown in Figure 4-10. Candidate aspects are identified analyzing the relationships between functional and nonfunctional concerns, which are then refined to a set of scenarios. Aspectual scenarios are represented as IPSs and non aspectual ones as UML Sequence Diagrams. The composition between aspectual and non aspectual scenarios is done by specifying instantiations of the IPSs which relate the IPSs to non aspectual scenarios, resulting in a new set of scenarios. In the next step, "Composing state machines", each scenario is translated into a set of state machines.

Name	the name of the nonfunctional concern
Description	informal description of the concern
Source	source of information (e.g. stakeholders, documents)
Decomposition	nonfunctional concerns can be decomposed into simpler ones. When all (sub) nonfunctional concerns are needed to achieve the nonfunctional concern, we have an AND relationship. If not all the sub nonfunctional concerns are necessary to achieve the nonfunctional concern, we have an OR relationship
Priority	expresses the importance of the nonfunctional concern for the stakeholders. A priority can be from 1 (very low) to 5 (very high)
Use cases	list of the use cases influenced by nonfunctional concern
Requirements	requirements describing the nonfunctional concern
Contribution	represents how a nonfunctional concern can be affected by the other nonfunctional concern. This contribution can be positive (+), negative (–) or none

Figure 4-10 Template for non functional concerns

The state machines from each scenario are then merged using a synthesis algorithm described in [26]. The result is a complete and executable state machine description of the requirements.
Regarding the translation to state machines, the first thing to do is convert each sequence diagram into a set of state machines, considering messages to an object as events and "replies" as actions, one for each object involved in the interaction. These state machines are then merged into a single state machine for that object, as shown in Figure 4-11.

In this approach, aspectual scenarios are represented as interaction pattern specifications (IPS). For each non aspectual scenario crosscut by them, a biding statement is given. These biding statements are used to instantiate aspectual scenarios.

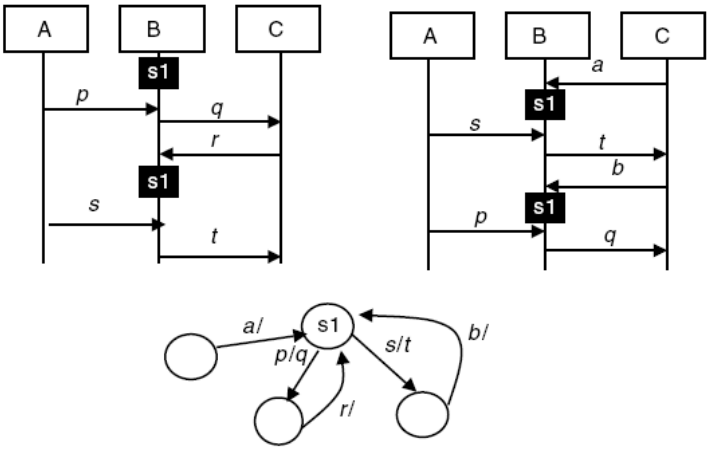

Figure 4-11 Synthesis from multiple sequence diagrams

The process of instantiation has three inputs, as shown in Figure 4-12, and result in a new sequence diagram that represents the weaving of I and S.

Elements	*Description*
Scenarios	An aspect IPS, *I*, and a sequence diagram, *S*, which *I* crosscuts
Binding	A binding of the role elements of *I* to concrete modelling elements in *S*
Integration operator	An integration operator, *op*, defining how *I* should be integrated with *S*

Figure 4-12 Inputs to instantiation of IPSs

There are three types of integration operators, illustrated in Figure 4-13:

- **OR:** specifies that I and S are alternative scenarios.
- **AND:** specifies that I and S should be executed concurrently.
- **IN:** specifies that I should be inserted in S.

To specify integration, one must find a "match" between the aspect and the concrete behavior.

By analyzing Figure 4-13, more precisely diagrams ***a*** and ***b***, a match can be found between messages "|*p*" and "*m2*" and between "|r" and "m5". An **OR** integration will be composed by the behavior specified by the concrete sequence diagram until the first match (m2), followed by an alternative behavior (*alt*) defined by the behavior specified in each of the considered diagrams until the second match.

An **IN** integration will correspond of a sequential behavior of all messages exchanged between the found "matches", as Figure 4-13, diagram *d* depicts.

When specifying an **AND** integration, in spite of an "alt" option, a "par" one must be considered.

After this weaving stage, the synthesis algorithm is applied and a set of state machines is generated, capturing all of the behavior specified by the original aspectual and non aspectual scenarios.

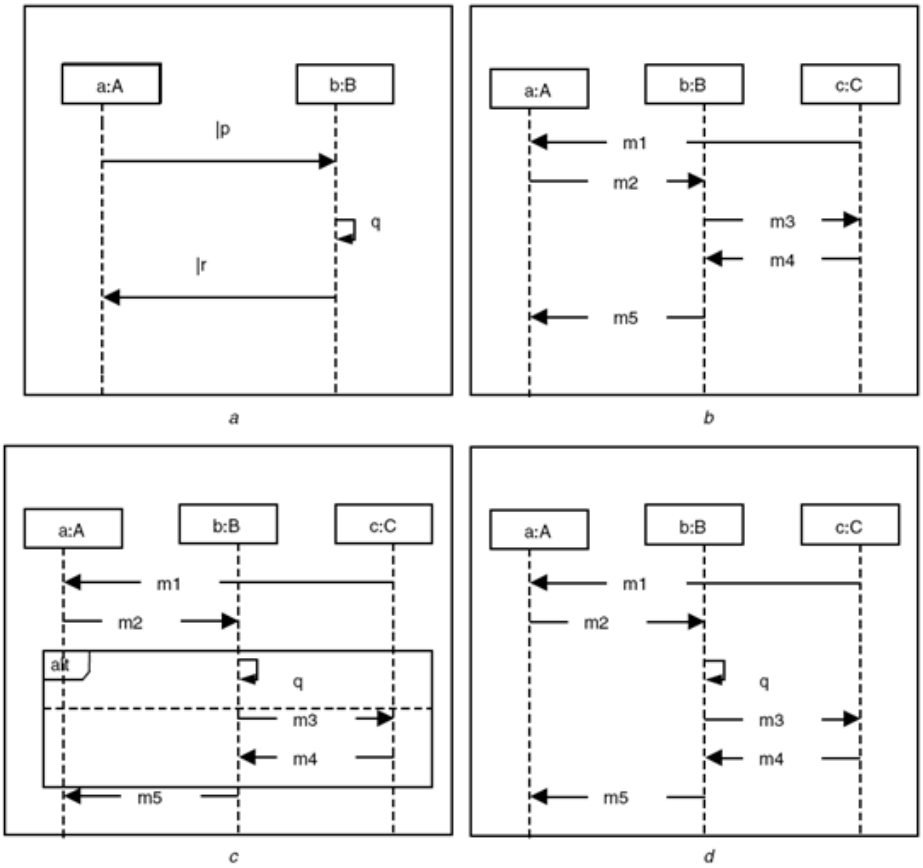

Figure 4-13 Instantiating an IPS aspect (a: IPS aspect; b: Concrete sequence diagram; c: OR integration; d: IN integration)

4.3.2 Modelling Aspects Using a Transformation Approach (MATA)

MATA [24], which is an evolution of the previously described approach, is an aspect-oriented modeling tool that considers aspect composition as a special case of model transformation.

Composition of a base model, M_b, with an aspect model, M_a, which crosscuts the base, is specified by a graph rule, r: LHS →RHS:

- A pattern is defined on the left-hand side (LHS), capturing the set of points in M_b where new model elements should be added;
- The right-hand side (RHS) defines those new elements and specifies how they should be added to M_b.

Currently, MATA only supports composition for UML class, sequence and state diagrams. MATA represents graph rules in UML's concrete syntax, however some minor extensions are introduced to allow for powerful pointcut and variable expressions. Furthermore, since writing graph rules using both a LHS and a RHS may imply writing repeated elements on both sides, a MATA rule is given in only one diagram, by using three new stereotypes:

- ***«create»:*** can be applied to any model element and specifies the creation of an element.
- ***«delete»:*** can be applied to any model element and specifies the deletion of an element.
- ***«context»:*** used with container elements, avoiding elements.

The application of these concepts is illustrated in Figure 4-14.

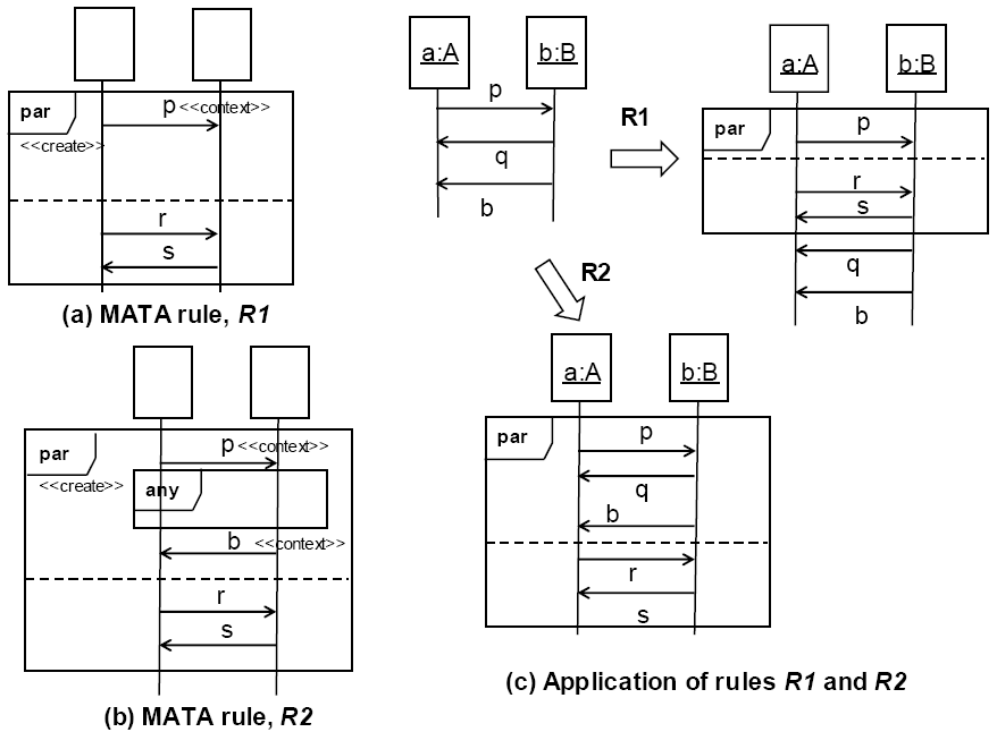

Figure 4-14 MATA rules

In order to detect dependencies and conflicts between rules a critical pair analysis (CPA) [12] can be used. Thus, if one rule requires a model element introduced by another rule, a CPA will declare a dependency. Conflicts are also detected if one rule modifies the base in such a way that another rule can no longer be applied; they can also mean that two rules that should both be applied cannot be, leading to the modification of these rules. These kinds of situations imply an ordering of the rules application, since the result may differ according to the first applied rule. However, if there are no conflicts or dependencies, then the rules can be applied in any order.

4.3.3 Aspect-Oriented Requirements with UML

This approach [5] aims at handling crosscutting non-functional concerns, i.e. global properties of a system that constrain the functional requirements, at the requirements stage. UML models were chosen as a notation to describe the requirements, thus this approach can be seen as a UML-based realization of the process presented in [19]. The process is composed by three main parts:

- ***Crosscutting concerns:*** the non-functional requirements are analyzed and crosscutting ones are identified, being described in a template, which specifies their name, description, priority, list of requirements and a list of models.
- ***Functional concerns:*** the functional requirements are specified via use case models, which are described using primary and secondary scenarios. Each scenario is further described using sequence diagrams.
- ***Composed requirements:*** functional requirements and aspects are composed and then possible conflicts are identified and resolved. The composition is defined in terms of overlapping (the aspect modified the functional requirements it transverses), overriding (the aspect behavior substitutes the functional requirements behavior) and wrapping (the aspect encapsulates the functional requirements it transverses).

Composition of crosscutting concerns may give rise to conflicts. These conflicts are treated in the same way as in the previously described approaches: if two concerns contribute negatively to each other, trade-offs are negotiated with stakeholders and a priority is attributed to each concern. The composition is then achieved considering that prioritization.

4.3.4 An aspectual Use-Case Driven Approach

Use cases are mostly used to define functional requirements, leaving out global properties that have some kind of impact in the system, thus the crosscutting nature of these requirements is not taken in account.

The approach presented here [4] extends the use case model to integrate non-functional requirements and to identify those use cases and NFRs that are crosscutting. The process is composed by five steps:

• ***Identify and Define Functional Requirements:*** analyzing the requirements, actors and use cases are identified, resulting in an initial use case model.

• ***Refine the UC model:*** functionalities that are included in more than one use case are externalized by using existing stereotypes («includes» and «extends»).

• ***Identify and Define NFRs:*** non-functional requirements are identified through the elicited requirements and from information obtained from the stakeholders. Each of these NFR is described in a template, specifying their name, description and a list of requirements.

• ***Integrate NFRs in UC model:*** the composition is achieved by extending the use case model via a new stereotype «constrain». Each non-functional requirement identified is stereotyped «NFR» and links between them and functional use cases are defined.

• ***Identify candidate aspects:*** if a use case constrains, extends or is included by more than one use case then it is crosscutting.

This approach can lead to a huge number of use cases, since a new use case for each NFR is added, besides all the use cases associated to functional requirements. To solve this, different use case diagrams can be built for each NFR, showing only the use cases directly affected by it.

4.3.5 Visualizing Aspectual Scenarios with Use Case Maps

The approach described in [17] explains how scenario-based aspects can be modeled at the requirements level and using the same techniques applied in non-aspectual systems with Use Case Maps. Use Case Maps are part of the User Requirements Notation (URN) [2] and define a notation based on a visual scenario language, which shows the interaction of architectural entities while abstracting from message and data details. Since UCMs integrate several scenarios and use cases into one combined model of a system, undesired interactions between scenarios can be analyzed. First, UCMs and aspects are unified via existing notational elements of UCMs to describe aspect-oriented concepts. The basic elements of the UCM notation are depicted in Figure 4-15.

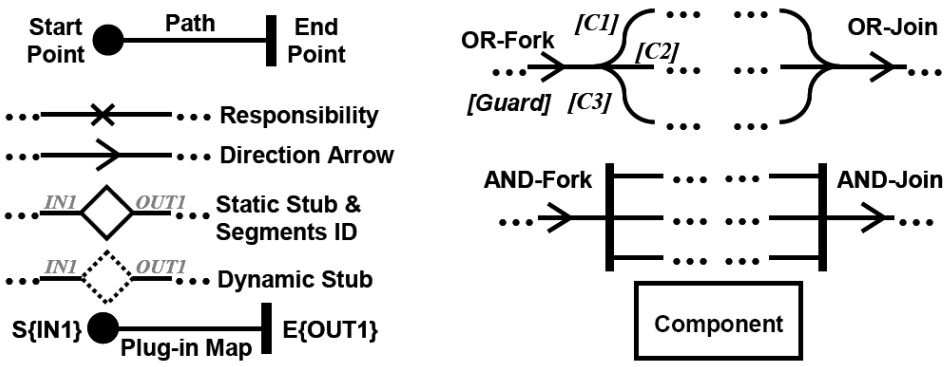

Figure 4-15 Basic elements of UCM notation

UCMs and UML activity diagrams can be considered similar, however UCMs are more flexible on how sub-diagrams are connected, sub-components are represented or how dynamic responsibilities and components are used to capture requirements for agent systems.

Associated with the notion of aspect are the following: *joinpoints*, which are locations in a program where behavior can be joined; *pointcuts*, which can be seen as a predicate that determines, for a given *joinpoint*, if it is matched by the predicate; and *advice*, which represents the behavior to be included in a *joinpoint*.

This approach only considers a UCM portion of the URN metamodel. Thus, in order to consider the new concepts introduced by aspects (advice map, pointcut map, pointcut stub and joinpoint), the URN metamodel needs to be extended. With the new classes an aspect can now be defined.

4.4 *Summary*

This Chapter has begun by showing the main application of scenarios, explaining some scenario based approaches.

Afterwards, the negative scenarios issue was introduced, with an explanation on their meaning and possible representation. Their relationships with "normal" use cases are also discussed.

Finally, some aspectual scenario-based approaches were introduced, namely MATA. This approach will be used as a composition mechanism between aspectual and base behavior, thus enhancing the Theme Approach by surpassing its lack in what concerns the referred mechanism. Since the notion of negative scenario (misuse case) was introduced, it makes sense to transpose this notion to the Theme Approach, including the concept of "misTheme", i.e., a theme representing an unexpected behavior, as what happens with the negative scenarios.

The effective use of these approaches on this work will be described in more detail in the next chapter.

Chapter 5
Modeling Aspectual Scenarios with Theme

Use cases are commonly used in Requirements Engineering due to its expressiveness - they are quite easy to understand by all stakeholders, from the most expert in the area to the final client who probably does not even know what a "requirement" means. However, and despite all the existent scenario based approaches, there is not a systematic way to identify them. The task of identifying use cases is extremely subjective: it depends on the analyst experience, thus it is possible, and very probable, for two different analysts to identify different use cases concerning the same system requirements.

On the other hand, starting with a set of requirements, the Theme Approach's first goal is to identify functionalities and crosscutting concerns (called aspects) of the system. This task is accomplished by analyzing the system requirements and by identifying the action words that potentially identify some functionality, each of them representing a theme. The approach also offers a set of heuristics to identify aspects.

However, when themes are refined into scenarios (as in Theme/UML), for example, the composition mechanism is not that expressive [24] since it is hard to have particular kinds of composition (e.g. parallel or conditional) as we can do using MATA.

We can see a use case model as a goal to be achieved by the developer when specifying a system. Therefore, we can take the advantage of the systematic way to find system properties present on the Theme Approach to obtain the use case model. Also, by applying the set of heuristics that the referred approach offers to identify aspects, we can also identify aspectual scenarios of the use cases. Having all the scenarios identified, aspectual and non aspectual, the next step would be to compose them. However, as previously said in Chapter 1, the composition mechanism offered by the Theme Approach is not expressive enough. To deal with this issue, a recent approach can be used: MATA, with its composition rules based on graph transformations.

However, considering that not only the expected situations happen, scenarios can also be used to illustrate them. Therefore, the notion of misuse case and negative scenarios will also be considered. Transposing this idea to Theme Approach results in a new concept: ***mistheme*** – an unexpected property that the system must handle. The whole system can then be composed taking into account not only what should happen but also contemplating possible unexpected behaviors.

Therefore, the proposed approach – ***MAST***, which stands for Modeling Aspectual Scenarios with Theme - takes the advantage of three existing approaches, extracting the best of them:

- ***Theme Approach:*** this provides a set of heuristics to find crosscutting behavior. Also, it provides a systematic manner to identify use cases, associating themes and its requirements to those use cases. This will make the scenario description easier;
- ***Use Case and Scenario-based approaches:*** easy to understand and provide stakeholders with an ample view of all important dynamic aspects of a system; they also let the analyst to express unexpected behavior and the way is affects the system;
- ***MATA:*** this provides an expressive composition mechanism.

The approach is depicted in Figure 5-1.

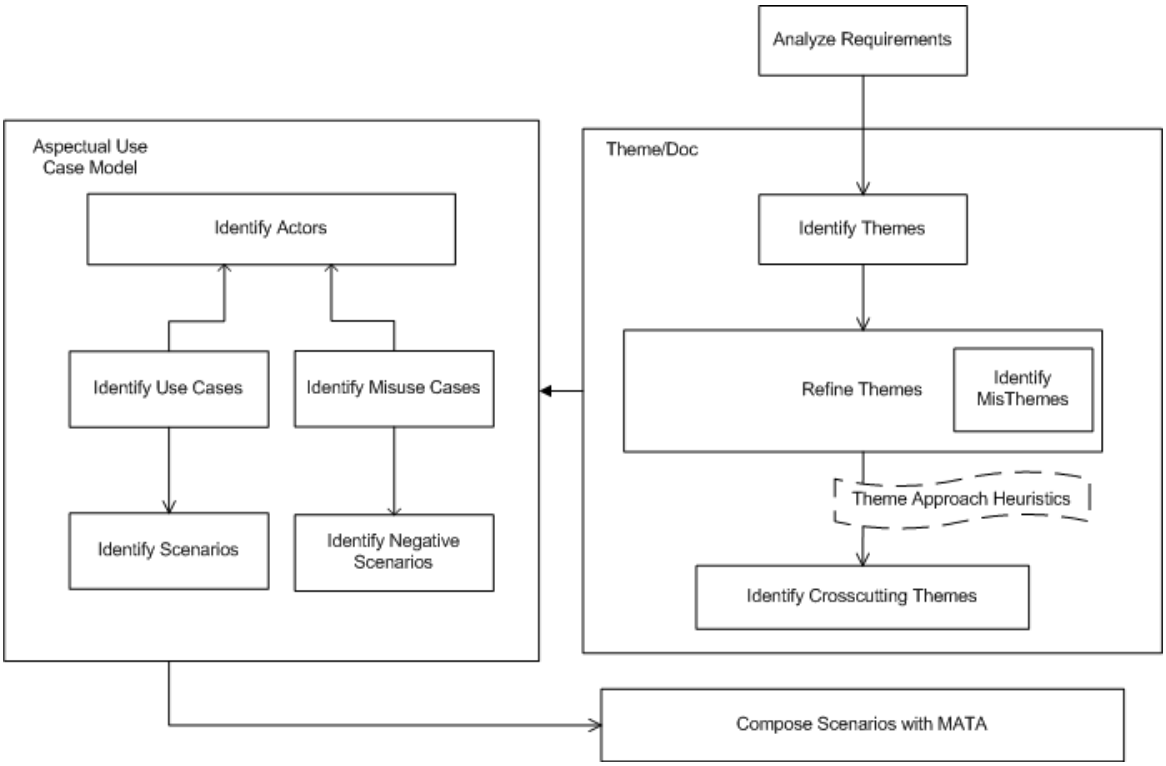

Figure 5-1 Process Model

The approach is constituted by two distinct models: Use Cases and Theme Approach, namely, the Theme/Doc portion.
Beginning with a set of requirements, we will take advantage of the Theme/Doc process:

- It assists in identifying the main themes on the specified requirements;
- The initial set of themes will be then refined, in order to achieve a coherent set of behavior;
- From this “final” set of themes, a set of misthemes, which identify unexpected situations, can be also identified;
- Finally, the Theme Approach heuristics will help in identifying crosscutting themes.

Having all themes identified, the Aspectual Use Case Model must be considered.
Each identified theme/mistheme will lead to a use case/misuse cases, respectively. Also, since each theme will be already identified as crosscutting or not, the correspondent use cases will also be identified as aspectual or not.
Associated with each use case/misuse case must be an actor, which is responsible for it. Each use case/misuse case will be described via scenarios/negative scenarios, which will be then composed via MATA.

5.1 *Applying the Approach to a Case Study*

The chosen case study is a simplified version of the toll collection system implemented in the Portuguese highways.

"In a road traffic pricing system, drivers of authorized vehicles are charged at toll gates automatically. These gates are placed at special lanes - the green lanes. To be recognized as an authorized vehicle, the driver has to install a device, called gizmo, on it. The registration of authorized vehicles includes the owner's personal data, bank account number and some vehicle details. When a vehicle passes on a toll gate, a sensor reads the gizmo. The information read is stored by the system and used to debit the respective account. When an authorized vehicle passes through a green lane, a green light is turned on, and the amount being debited is displayed. If an unauthorized vehicle passes through it, a yellow light is turned on and a camera takes a photo of the plate. The amount a drivers needs to pay depends on the type of the vehicle and the distance travelled."

5.1.1 Analyzing Requirements

The first step in the Theme Approach consists of analyzing the requirements in order to identify themes.
The requirements must respect the following syntax: ***(Condition | Subject) + Action + Object***. Thus, all the requirements must start with a Condition or Subject, following by an Action and an Object.

- R1) The vehicle owner must register its vehicle, associating it with a gizmo. A communication with the bank must be established in order to guarantee a good account.

- R2) The vehicle passes through single toll gates.

- R3) A registered vehicle gizmo must be read and validated when entering or exiting the toll gates.

- R4) If the gizmo is a valid one, turn a green light on.

- R5) If the gizmo is not a valid one, turn a yellow light on and take a photo.

- R6) When a vehicle enters a toll gate, register the entrance.

- R7) If the vehicle tries to exit a toll gate, read and verify the gizmo and verify if the vehicle has an entrance.

- R8) If the vehicle did not enter in a green lane, turn a yellow light on and take a photo.

- R9) Sum up all the passages for each vehicle; a debit is sent to the bank and a copy of it to the vehicle owner.

- R10) When the vehicle tries to exit a toll gate, calculate and display the amount to be paid.

- R11) A gizmo must be registered and associated with an account in order to be considered valid.

5.1.2 Identifying Themes

Analyzing all the action words present on requirements, the following potential themes were identified:

- T1) Register vehicle
- T2) establish communication
- T3) read gizmo
- T4) Validate gizmo
- T5) Turn on light
- T6) Calculate amount
- T7) Display amount
- T8) Pass toll gate
- T9) Take photo
- T10) Enter Toll Gate
- T11) Register entrance
- T12) Verify entrance
- T13) Exit Toll Gate

- T14) Sum up passages

- T15) send debit

5.1.2.1 Refining Themes

A Theme, as previously stated, can be seen as a "coherent set of behavior", defining a subset of responsibilities. So, the first two Themes can be grouped, since the establishment of communication between the bank and the gizmo's user, referred on Theme 2, only makes sense in the context of a vehicle registration.
Themes 3 and 4 always happen sequentially.
Themes 5 and 9 only happen after Theme 4, thus the three themes can be composed. For the same reason, Themes 6 and 7 are merged. Themes 10 and 11 deal with entrances, thus it makes sense to group them.
Finally, Themes 14 and 15 represent a bill action, thus it makes sense to treat them as a single functionality.

Once these themes are grouped, a *relationship view* can be defined.
A relationship view shows relationships between themes and the requirements that justify each theme existence. Each theme is represented as a diamond-shaped node and requirements are identified as boxes. Each theme and requirement should be labeled with its text or by a number.

Thus, to build a relationship-view, one has to identify all the requirements that must be considered in order to achieve the functionality described by a theme. For instance, Theme 1 is defined as "register vehicle" - the only requirement that has something to do to a vehicle registration is R1, thus these two elements must be linked. On the other hand, regarding Theme 12: R7 explicitly refers the "Verify Entrance" action; R8 and R3 also happens on a "Verify Entrance" context, since in order to verify an entrance, the gizmo must be read and validated and, if an entrance has not been registered, a yellow light must be turned on and a photo taken – thus, Theme 12 must be linked with R3, R7 and R8.

A relationship view regarding the described system is shown in Figure 5-2 Relationship viewFigure 5-2.

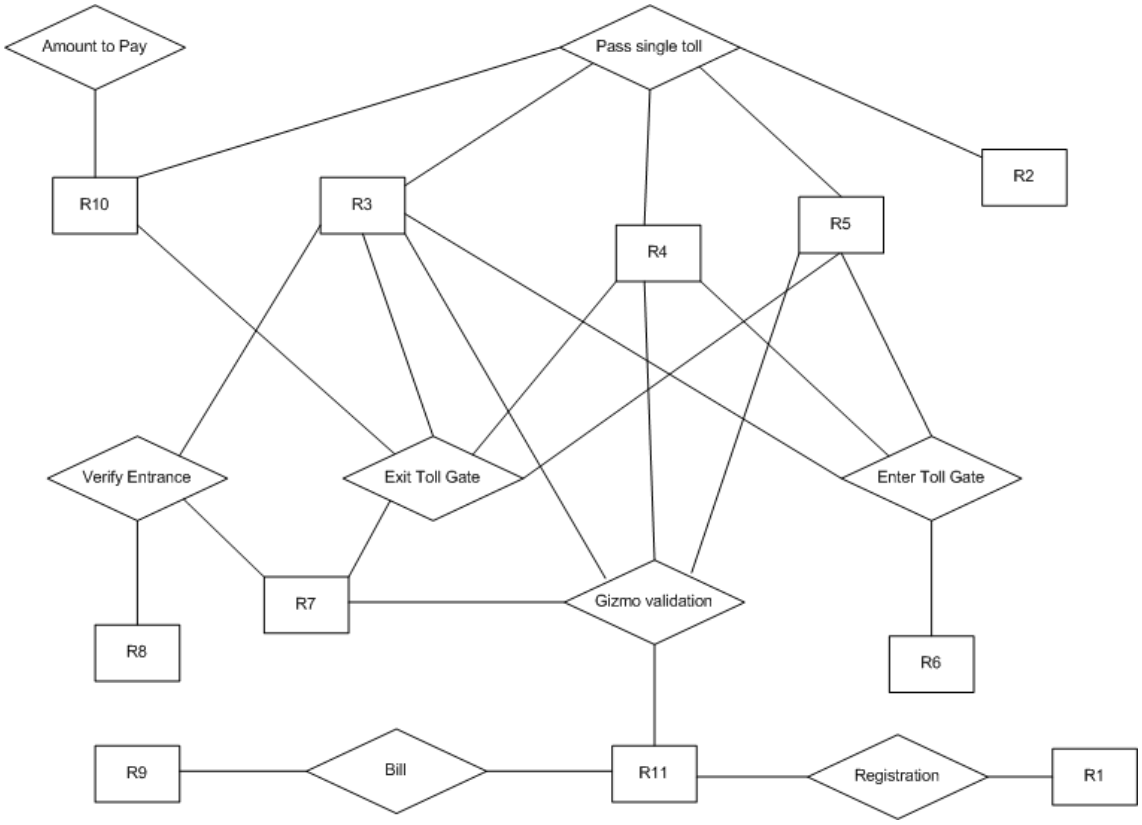

Figure 5-2 Relationship view

At this point, all the found themes seems to make sense as an understandable functionality of the system, namely to deal with "vehicle registration", "pass toll gate", "amount to pay", "gizmo validation", "enter toll gate", "verify entrance", "exit toll gate" and "bill".

5.1.2.2 MisThemes

We can look to the found themes and make a question: "And what if this action does not occur? And if the contrary action does happen?"- If we can find an answer for this question, a negative theme – ***mistheme*** – was found.
Having this in mind let's analyze the found themes:

T1) Register vehicle

What should happen if a vehicle has not been registered? The registration is a necessary action in order for a vehicle driver to have a gizmo. Thus, if one does not register its vehicle will not have a

gizmo. And what should happen if a vehicle driver without a gizmo tries to pass through a toll gate? In this case, a yellow light must be turned on and a photo must be taken in order to register an infraction – we have found the first mistheme.

- MT1) Pass without gizmo

This mistheme is carried out by an **offender driver** and is closely related to three themes: "Pass Single Toll", "Enter Toll Gate" and "Exit Toll Gate", since all of them suppose that a gizmo is present.

What about the "Gizmo Validation" theme? What should happen if a vehicle driver passes through a toll gate with an invalid gizmo? What should happen if a gizmo was stolen and an unauthorized vehicle driver tries to pass through the toll gates with that gizmo? A new infraction must be registered, a yellow light must be turned on and a photo taken.

- MT2) Invalid gizmo

There are three themes left: "Bill", "Amount to pay" and "Verify Entrance".

If we take a look into those themes' related requirements we can conclude that there are not any issues related to both "Bill" or "Amount to pay" themes, but what should happen when a vehicle driver passes through a exit toll gate and an associate entrance does not exist or, for some reason, such as the gizmo has not been identified as a valid one, was considered invalid? Again, an infraction must be registered, a yellow light turned on and a photo taken.

- MT3) Invalid Entrance

The idea behind misthemes identification is to know how to deal with unexpected situations originated from a bad intentional behaviour of an "Offender" actor, thus the system must specify what should happen if something goes wrong when someone wants to violate the system, i.e., there should exist a (set of) requirement(s) that define what to do in this kind of situations.

If we look this system's requirements and verify what should happen on each of the described situations, we can conclude that we already have requirements to deal with the identified unexpected behavior related to misthemes 2 and 3, namely

- *R5) If the gizmo is not a valid one, a yellow light is turned on and a photo is taken.*

and

- *R8) If the vehicle did not enter in a green lane, a yellow light is turned on and a photo is taken.*

However, there is not a requirement that could handle a "pass without gizmo" situation. In this case, a new requirement must be created. Requirements created in order to deal with unexpected situations will be considered ***negative requirements (NR)***. Thus, in this particular situation,

- NR1) If a gizmo is not present, a yellow light is turned on and a photo is taken.

All these misthemes were identified by analyzing all the previously found themes, thus these two kinds of themes should be related, since the first were identified from the second ones.

Also, there are several shared requirements between themes, being not clear which theme should be responsible for which behavior. The next step is to address the relationships between these themes to determine if one theme dominates another concerning its functionalities.

5.1.3 Identifying Crosscutting Themes

As previously referred, the process of identifying crosscutting themes begins with a set of themes. To identify aspects, one needs to find themes that share a requirement.
If the requirement cannot be split up and if one theme is dominant in the requirement, then the theme should be responsible for the requirement. Moreover, if the behavior of the dominant theme is triggered by other themes mentioned in the requirement, then there is a trigger relationship between two themes. Finally, if the dominant theme is triggered in multiple situations, then it is crosscutting and becomes an aspect.

We have seven shared requirements: R3, R4, R5, R7, R10 and R11.

R3 is linked to "Gizmo Validation", "Enter Toll Gate", "Pass single toll", "Verify entrance" and "Exit toll gate". The requirement definition explicitly refers a gizmo validation, thus it is clear that the "Gizmo Validation" theme is the dominant theme in the requirement and should be responsible for it. Moreover, the "Gizmo Validation" is triggered by "Enter toll gate", "Pass single toll" and "Exit toll gate" themes, thus defining a triggering relationship with them.

R4 is linked to "Exit toll gate", "Gizmo validation", "Enter toll gate", "Pass single toll". The requirement definition shows that the "Gizmo validation" is the dominant theme, since it is a direct consequence of a gizmo validation, thus being responsible for it, and that is triggered by the other themes mentioned in the requirement, composing a triggering relationship between them.

R5 is linked to "Exit toll gate", "Gizmo validation", "Pass single toll". For the same reason as before, "Gizmo validation" is the responsible theme for the requirement, being triggered by the "Exit toll gate" and "Pass single toll".

R7 is shared by two themes: "Exit toll gate" and "Gizmo validation", being the former the theme responsible for it.

R10 is linked to "Exit toll gate", "Pass single toll" and "Amount to pay", being the last theme responsible for it. Also, the behavior defined by the “Amount to pay” theme is triggered in multiple situations: when exiting a toll gate or when passing through a single toll.

Finally, R11 is linked to “Gizmo Validation”, “Bill” and “Registration”. The requirement mostly defines what is needed for a gizmo to be valid, thus either “Bill” or “Registration” could be responsible for it. However, and despite of not being clear which of them should be the responsible theme, there is not a triggering relationship.

Analyzing all the dominant themes and triggering relationships, one can conclude that "Gizmo validation" and “Amount to pay” are triggered in multiple situations, thus being classified as crosscutting themes.

Now we can relate all the found themes. Thus, the following steps will only detail the already identified relations.

All the found themes would compose the whole system, thus all of them should be connected. Therefore, all the found themes can be related to each other having in mind the existent dependencies between the functionalities that each theme describes. The identified triggering relationships should also be taken in account. Theme relationships are a contribution of this work, not present in the original version of Theme.

To represent a triggering relationship between two themes, an arrow must link the two themes, starting on the triggered behavior (aspect) and ending on the theme that represents the base – an **"<<*affects*>>"** stereotype should be used.

The behavior described by one theme may also be required by another in order for that theme's goal accomplishment. To represent this kind of dependency a **"<<*requires*>>"** stereotype must be used, starting on the theme that requires some behavior and ending on this one.

On the other hand, a theme's behavior may be used by another one – to represent this kind of relationship a **"<<*may use*>>"** stereotyped is available, connecting the theme that may use other theme's behavior to that theme.

This dissertation's contribute is to take advantage of the synergy between two complementary approaches: the Theme Approach and Use case/Scenario-based approaches. Thus, additionally, we can relate the existing relationships that each of these approaches defines to better define the proposed relationships according to this dissertation's approach.

The Theme Approach relates the found themes with a crosscutting relationship; Use Cases, on the other hand, let one to specify if one use case can include or extends another one.
Therefore, according to Theme, we can specify if one theme is crosscutting or not; however, use cases give us the knowledge to identify that crosscutting as mandatory or optional, if, besides the crosscutting behavior of some theme, that same behavior is required or may be used by another, respectively.

Table 1 intends to illustrate the previously defined concepts.

Table 1 Themes vs. Use Cases: Relating stereotypes

Themes	Use Cases
<<*requires*>>	<<*includes*>>
<<*may use*>>	<<*extends*>>
<<*affects*>>	<<*includes*>> or <<*extends*>> (if one use case is included by more than one use case or if it extends more than one use case, respectively)

Figure 5-3 depicts a mandatory crosscutting relationship between two themes.

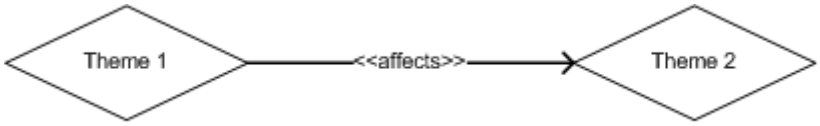

Figure 5-3 Mandatory crosscutting

Figure 5-4illustrates an optional crosscutting relationship between two themes.

Figure 5-4 Optional crosscutting

This step defines the first point in which two complementary approaches begins to relate with each other.

Now, let's take a look at the found themes: there are some obvious relations – in order to exit a toll gate, an entrance must be verified, and thus "Exit Toll Gate" and "Verify Entrance" should be linked, via a **"<<*requires*>>"** stereotype.

When a user exits a toll gate or passes through a single toll, the amount to be paid is calculated and displayed – both "Exit Toll Gate" and "Pass Single Toll" must be linked with "Amount to pay".

The "Gizmo Validation" is a necessary action if a user wants to "Enter a Toll Gate", "Exit a Toll Gate" or "Pass a Single Toll". Thus, a link must be established between the first theme and the following. Also, it was identified a triggering relationship between "Gizmo Validation" and the other mentioned themes. Thus, the links must take the form of an arrow, with an **<<*affects*>>**

stereotype, starting in the "Gizmo Validation" and ending in each of the other themes. This relationship also identifies a *mandatory crosscutting*.

Finally, a "Gizmo validation" only makes sense if some kind of connection already exists between the user and the gizmo. Therefore, the "Gizmo validation" theme should be linked to the "Registration" theme with a ***"<<requires>>"*** stereotype.

We have also identified a set of misthemes, themes which behavior counteracts other themes' behavior, and have argued that a mistheme should be related to the theme(s) from which it was identified.

A mistheme can be represented the same way as a "normal" theme, also assuming a diamond shape. However, in order for them to be easily distinguished, the background of the mistheme's diamond should be darker.
To show that a mistheme behavior is transgressing a "normal" theme's behavior, a ***"<<transgresses>>"*** stereotype should be used.
Figure 5-5 depicts all the found themes and their relationships.

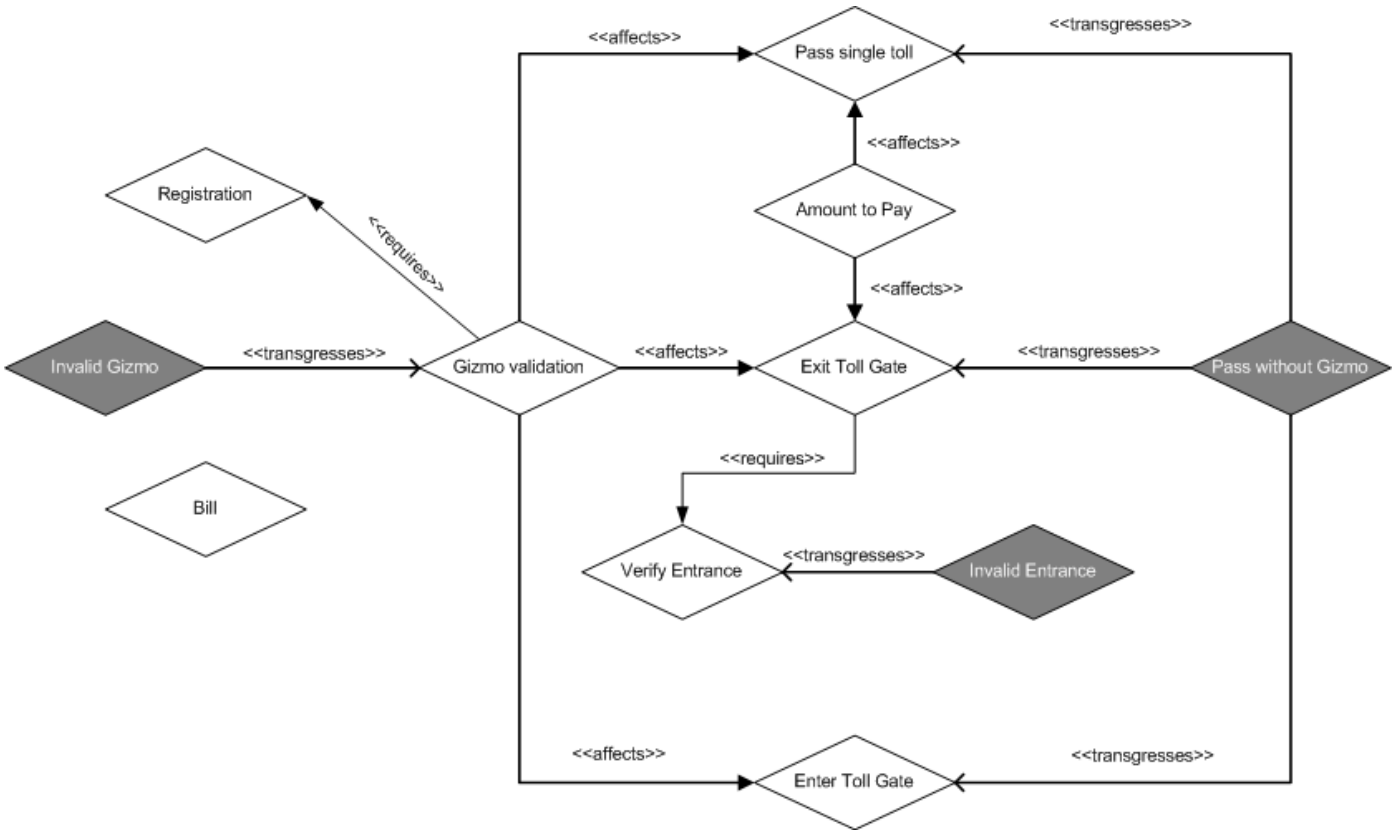

Figure 5-5 Relating Themes

5.1.4 Identifying Actors, Use Cases, Misuse Cases and Scenarios

A theme/mistheme represents a set of coherent behavior that represents a functionality of the system. And so represents a use case. Thus, each theme/mistheme identifies a use case/misuse case, respectively. Related with each use case is some actor.

The following actors have been identified, analyzing also the requirements:

- **Vehicle owner:** responsible for registering a vehicle;
- **Vehicle driver:** identifies the vehicle, the driver and the gizmo installed on it;
- **Bank:** entity that holds the vehicle owner's account;
- **System clock:** internal clock of the system that monthly triggers the calculation of debits.

To identify which actor(s) is(are) related to each use case one needs to ask who is responsible for each functionality.

Through the definition of each requirement, it is possible to identify if some use case should include or extends another one. A “must”, “need to” or “have to” expression on a requirement definition shows that the requirement under analysis should include another one, which can be identified on the remaining definition of that requirement.

Again, let’s take a look at each theme/use case.

A “Registration” is something that the “Vehicle owner” must do.

The “Bill” issues are in charge of the “Vehicle owner”, who provides his personal data to a bank entity in order to process all the necessary debits, and also to a generic entity “Clock”, responsible to monthly sum up all the passages and sent them to the bank and to the user.

The “Vehicle driver” is who *enters/exits Toll Gates* or *passes through single tolls*.

The g*izmo validation* does not belong to a specific actor: it defines an action that must be carried out on each passage, according to R3 definition, thus it is included on “Enter Toll Gate”, “Pass single Toll” and “Exit Toll Gate”. The *amount to be paid* is only calculated when a vehicle driver *passes through a single toll* or *exits a toll gate*, also defining a necessary action; thus the “Amount to pay” use case should be included by both “Pass Single Toll” and “Exit Toll Gate”. Finally, an entrance “must be verified” when a drivers exits a toll gate, being then included by it.

Regarding the found misthemes:

- the “Invalid Gizmo” mistheme describes a situation in which an unauthorized vehicle driver tries to pass through the toll gates with a stolen gizmo or tries to pass even knowing that the gizmo is invalid (battery discharged, for instance);
- the “Invalid Entrance” is also carried out by an offender driver who, when enters on a motorway, do not pass through a green lane;
- “Pass without gizmo” is an action that is carried out by an unauthorized vehicle driver;

In order to distinguish a vehicle driver who obeys to all rules from another who is responsible for some infraction, a new actor should be considered – **“Offender driver”**.

The identified associations are depicted in Figure 5-6.

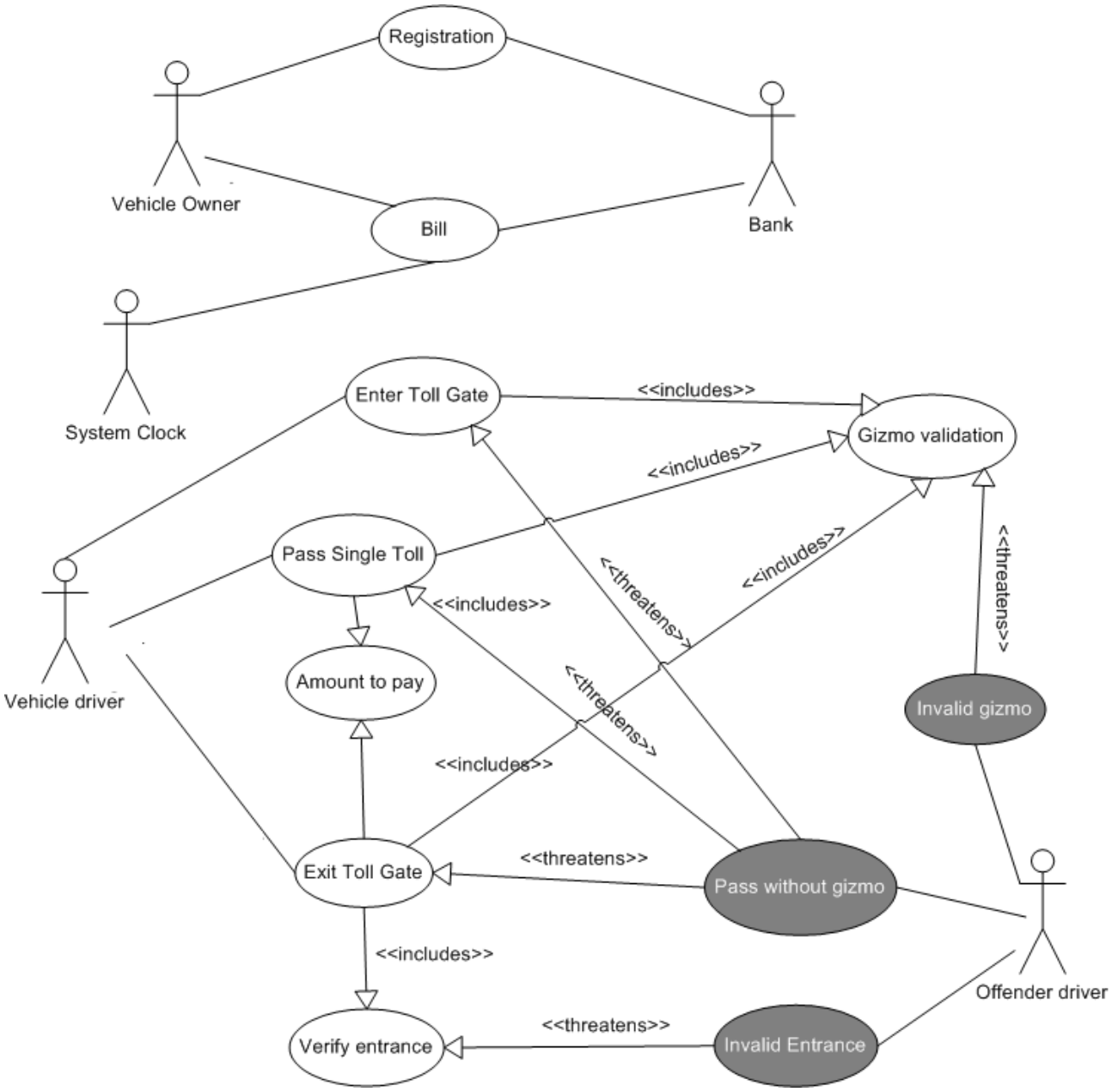

Figure 5-6 Use Cases diagram

At this point we have already identified requirements, themes and use cases/misuse cases: starting with a set of requirements, we have extracted all the actions words, identifying a set of potential themes. These set of themes have been refined and a final set of themes were considered. Analyzing those themes, a set of misthemes were also identified. Each theme/mistheme, being seen as a coherent set of behavior, has also identified a use case/misuse case.

So, each identified theme will lead to a use case; therefore the related scenarios can be identified by "grouping" all the requirements linked to that theme. This is possible due to the required syntax for

the found requirements. The grouped requirements will result on a description of that use case – a requirement or a set of requirements express a use case scenario.

Regarding the identified misthemes, a set of misuse cases have also been identified. A mistheme is identified by finding some transgressing behavior regarding some theme's behavior. So, in order to describe a misuse case related scenario, one must analyze the behavior that the use case identified by that theme describes – the use case's scenario. On that scenario, one have to identify the step in which the transgression is possible. The misuse case's scenario will consist on that step description. The relationships between all these concepts, as well as a description of the system's scenarios, are shown in Table 2and Table 3.

Table 2 Themes, Use Cases and Scenarios

Themes ←→ Use Cases	**Scenarios**
Gizmo validation	R3) A registered vehicle gizmo must be read and validated when entering or exiting the toll gates.
	R4) If the gizmo is a valid one, turn a green light on.
	R5) If the gizmo is not a valid one, turn a yellow light on and take a photo.
	R11) A gizmo must be registered and associated with an account in order to be considered valid.
Verify Entrance	R3) A registered vehicle gizmo must be read and validated when entering or exiting the toll gates.
	R8) If the vehicle did not enter in a green lane, turn a yellow light on and take a photo.
Registration	R1) The vehicle owner must register its vehicle, associating it with a gizmo. A communication with the bank must be established in order to guarantee a good account.
	R11) A gizmo must be registered and associated with an account in order to be considered valid.
Enter Toll Gate	R3) A registered vehicle gizmo must be read and validated when entering or exiting the toll gates.
	R6) When a vehicle enters a toll gate, register the entrance.
Exit Toll Gate	R7) If the vehicle tries to exit a toll gate, read and verify the gizmo and verify if the vehicle has an entrance.
	R10) When the vehicle tries to exit a toll gate, calculate and display the amount to be paid.
Pass Single Toll	R2) The vehicle passes through single toll gates.
	R3) A registered vehicle gizmo must be read and validated when entering or exiting the toll gates.
	R10) When the vehicle tries to exit a toll gate, calculate and display the amount to be paid.
Amount to pay	R10) When the vehicle tries to exit a toll gate, calculate and display the amount to be paid.

Bill	R9) Sum up all the passages for each vehicle; a debit is sent to the bank and a copy of it to the vehicle owner.
	R11) A gizmo must be registered and associated with an account in order to be considered valid.

Table 3 Misthemes, Misuse Cases and Negative Scenarios

MisThemes ←→ Misuse Cases	**Negative Scenarios**
Invalid Gizmo	R5) If the gizmo is not a valid one, turn a yellow light on and take a photo.
Invalid Entrance	R9) If the vehicle did not enter in a green lane, turn a yellow light on and take a photo.
Pass without gizmo	NR1) If a gizmo is not present, a yellow light is turned on and a photo is taken.

5.1.5 From Aspects to Aspectual Scenarios

Starting with a list of requirements, the Theme Approach has helped in identifying a set of functionalities related to the described system. Also, by providing a set of heuristics, Theme has let one to identify crosscutting behavior. Each found functionality identifies a theme. All functionalities were found through the requirements analysis, thus, related to each theme, will be one or more requirements. Also, each theme will lead to a use case.

A use case, being also seen as functionality, can be described via scenarios: a set of steps that must be done in order to achieve the use case's goal. Thus, if a theme identifies a use case and is related to some requirements, then that use case's scenarios can be found through the requirements that are linked with the corresponding theme. Seeing that a use case can be classified as crosscutting or not, depending if the related theme is classified as an aspect or not, a scenario can also be identified as aspectual or non-aspectual. The same goes for misthemes and correspondent misuse cases and negative scenarios.

Hence, the next step will result on the composition of base and aspectual scenarios in order to compose the system as a whole and not just as a set of parts.

5.1.6 Refining and Composing the Scenarios with MATA

Now that all the different concepts have been associated, the next step is to compose the scenarios of each use case. Until now, the Theme approach has helped with the requirements analysis, introducing a set of heuristics which are helpful to recognize aspectual behavior through the system requirements. The problem with the approach concerns this step: the composition.

The Theme approach defines a composition has a "dashed line between the design elements to be composed" [7]; on this line a tag can be specified, defining the new theme name. An example of this kind of representation can be seen in Figure 5-7.

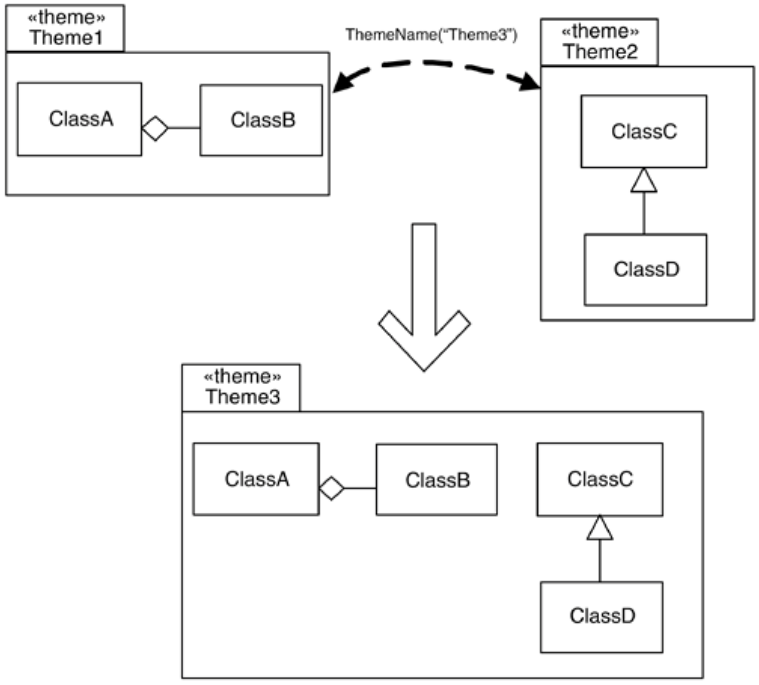

Figure 5-7 Composing Themes on Theme Approach

On Theme Approach, themes can also be composed by using a "merge integration", which results on the union of the models in the input themes. Again, a tag should be used to specify the composition. And the same happens in multiple situations: on override Integrations, on theme precedence definitions, when specifying bindings to aspect themes. It is easy to conclude that Theme Approach's composition mechanism is not that expressive: in order to understand this entire notation, one needs to learn all the composition process, with all these tags and meanings. Also, it is not possible to clearly specify, where, in the base, the aspectual behavior should occur: the available operators are based on AspectJ and, thus, only provides the "after", "before" and "around" ones.

In this context, emerge the found scenarios.

As previously stated, MATA constitutes an expressive composition mechanism, used to compose scenarios. Therefore, one can use the found scenarios on this approach to define the desired composition for those themes with MATA. There are two distinct models in MATA: the base model and the aspectual one, being this one included on the previous through the use of some stereotypes, namely:

- **<<create>>:** adds an element to the base;
- **<<delete>>:** removes an element from the base;
- **<<context>>:** defines the context in which the aspect occurs.

The Theme Approach has already defined which of the themes are considered as base or aspect, which in turn has lead to base and aspectual scenarios identification. Therefore, there are two crosscutting scenarios: "Gizmo validation" and “Amount to pay”. These scenarios affect, respectively, "Enter Toll Gate", "Pass single Toll" and "Exit Toll Gate", and “Exit Toll Gate” and “Pass Single Toll”, which are the Base scenarios.

The crosscutting behavior can be composed with the non-aspectual one by considering the following considerations:

- When a crosscutting behavior is triggered by an event, it occurs in response of some action, thus it happens on a certain context;
- When a crosscutting behavior is included on the base one, it adds or removes elements on it.

Taking advantage of the Theme Approach heurists, one could find a systematic way to identify themes, which can be used to identify the correspondent use cases.
A scenario, as an instance of a use case, represents a single use case interaction. This may be a main flow or another possible variation, consisting of a set of actions that must happen in order to achieve certain goal - the use case goal.
To identify a use case scenarios, and taking in account that a theme identifies a use case and the relationship view that links themes with their requirements, one only need to “organize” each use case requirements, defining a coherent sequence of actions that must be carried out in order to achieve certain goal.

A sequence diagram shows interactions among objects that live simultaneously. All the messaged exchanged between them are represented horizontally, in the order in which they occur. This allows the specification of scenarios in a graphical manner.

Thus, each of the identified scenarios can then be detailed on sequence diagrams. Each sequence diagram will show how base and crosscutting behavior can be composed. To define this composition, the set of MATA stereotypes should be used, establishing a connection with the diagrammatic composition with MATA, via sequence diagrams.

But before specifying the sequence diagrams it is better to design them first.
As previously stated, a scenario can be represented as a set of actions and/or conditions. Thus, each scenario will be described as a textual description of all the actions and/or conditions that have been described on that scenario's requirements. The result of each condition will be described as "RTN someCondition". The global description follows an algorithm-based style.

For instance, considering the aspectual scenario "Amount to pay" and the MATA notation:

- The scenario happens when a vehicle driver passes through a single toll and the gizmo is validated or, when trying to exit through a toll gate, a correspondent entrance is verified: this defines the scenario context and should be stereotyped as "<<context>>";
- The scenario itself is defined as a single action (calculate amount to be paid): this is the aspectual behavior that should be inserted on the base, thus it is stereotyped as "<<create>>".
- Between the context and the action that must be inserted on the base may occur another action, as well as after it - this is represented as "***/*do something */***".

The described scenario can then be represented in MATA as:

*<<**context**>> (ValidateGizmo(gizmo) || VerifyEntrance(gizmo));*
/ do something */*
*<<**create**>> CalculateAmountToPay();*
/ do something */*

Now, let's see how this aspectual behavior can be composed with the base.

We have already identified which bases the “Amount to pay” aspect affects: “Exit Toll Gate” and “Pass single Toll”.

A base scenario does not include any behavior on an aspectual one: it only triggers the aspectual behavior, thus it can be represented a sequence of actions.

Therefore, the “Exit Toll Gate” would be refined into:

DetectGizmo();
ValidateGizmo(gizmo);
VerifyEntrance(gizmo);
RegisterExit(gizmo).

On the other hand, the “Pass Single Toll” can be defined as:

DetectGizmo();
ValidateGizmo(gizmo);
RegisterPassage().

To compose aspectual behavior with the base in which it occurs, one need to find, in the base description, an action that matches with the action that defines the aspect context.

The “Amount to pay” context is defined as “ValidateGizmo” or “VerifyEntrance”. Looking at the “Exit Toll Gate” description, we have a match with the second and third actions, *ValidateGizmo(gizmo)* and *VerifyEntrance(gizmo).* Although a “ValidateGizmo” also occurs on the base description, the match should be done with the most specific action regarding the goal to be achieved on that scenario, thus with *“VerifyEntrance(gizmo)”* - the aspectual behavior should be included on that point, defining the following composed scenario only considering the “Amount to pay” behavior – note that the “Exit Toll Gate” is also affected by the “Gizmo Validation” aspectual behavior.

DetectGizmo();
ValidateGizmo(gizmo);

VerifyEntrance(gizmo);
CalculateAmountToPay();
RegisterExit().

We have also identified negative scenarios, namely “Invalid Gizmo”, “Invalid Entrance” and “Pass without gizmo”. A composed scenario will also contain a negative part. To identify where a negative scenario should be included, one have to:

1) Identify which scenarios can be affected by a negative one;
2) Identify, on the previously identified scenarios, where a negative scenario may occur;
3) Include the negative scenario description on the composed one.

A negative scenario identifies an unexpected situation. On the other hand, when defining what is supposed to happen on a certain system’s context, only the “normal” situations are considered. Hence, while a use case defines an “if condition = true” situation, a misuse case specifies the contrary situation – “if condition = false”, thus an “else”.

Thus, concerning the “Exit Toll Gate” scenario, the Use Cases diagram (see Figure 1.4) tell us that a “Pass without gizmo” may occur, which, according to the correspondent scenario (see Figure C) means that “If the gizmo is not present a yellow light is turned on and a photo is taken”.
Also, an “Invalid Gizmo” may occur, associated with a “Gizmo validation” action, thus “If the gizmo is not a valid one, a yellow light is turned on and a photo is taken.”

When a vehicle drivers exits through an exit toll gate, the gizmo is detected and validated and, if it is valid, is verified if there is some valid entrance. If this is the case, the amount to pay is calculated, showed to the vehicle driver and an exit is registered. However, if the gizmo is not a valid one, a negative scenario should be considered - the “Invalid Gizmo”. Also, if there is no valid entrance, another negative scenario arises – “Invalid Entrance”.

Concluding, when considering a single use case, it is not enough to only consider all the direct misuse cases: one should also analyze all the related (included or extended) use cases and, for each, consider the possible misuse cases.

Having this in mind, the whole “Exit Toll Gate” scenario will take in account three negative scenarios assuming the form:

```
DetectGizmo();
if (RTN DetectGizmo() == OK)
        ValidateGizmo(gizmo);
        if (RTN ValidateGizmo(gizmo) == OK)
                VerifyEntrance(gizmo);
                if(RTN VerifyEntrance(gizmo) == OK)
                        CalculateAmountToPay();
                        RegisterExit()
                else
                        TurnOnLight(yellow)
                        TakePhoto()
        else
                TurnOnLight(yellow)
                TakePhoto()
else
        TurnOnLight(yellow)
        TakePhoto()
```

Concerning the "Pass single toll" behavior, we have a match on the second action, "*ValidateGizmo(gizmo)"*, which identifies where the aspectual behavior should occur.

Also, directly related to the "Pass single toll" use case is the "Pass without gizmo" misuse case. However, when passing through a single toll, the gizmo validation also occurs, and thus the "Invalid Gizmo" negative scenario should be also considered.

A description of the composed scenario (note that the "Pass single toll" is also affected by the "Gizmo Validation" aspectual behavior) will take the form

```
DetectGizmo();
if (RTN DetectGizmo() == OK)
        ValidateGizmo(gizmo);
        if (RTN ValidateGizmo(gizmo) == OK)
                CalculateAmountToPay();
```

RegisterPassage()

else

TurnOnLight(yellow)

TakePhoto()

else

TurnOnLight(yellow)

TakePhoto()

The negative scenarios can be seen in Listing 1, Listing 2 and Listing 3.

Invalid Gizmo	`if ValidateGizmo(gizmo) != OK`
	`  TurnOnLight(yellow);`
	`  TakePhoto().`

Listing 1 Negative Scenario - Invalid Gizmo

Invalid Entrance	`if VerifyEntrance(gizmo) != OK`
	`  TurnOnLight(yellow);`
	`  TakePhoto()`

Listing 2 Negative Scenario - Invalid Entrance

Pass without gizmo	`if DetectGizmo(gizmo) != OK`
	`  TurnOnLight(yellow);`
	`  TakePhoto().`

Listing 3 Negative Scenario - Pass without gizmo

Listing 4 shows an aspectual scenario found.

Gizmo Validation	`<<context>> DetectGizmo();`
	`if( RTN DetectGizmo() == OK)`
	`  <<context>> ValidateGizmo(gizmo);`
	`  if(RTN ValidateGizmo(gizmo) == OK)`
	`  <<create>> VerifyState(gizmo);`
	`  if (RTN VerifyState(gizmo) == active)`
	`  /* do something */`
	`  TurnOnLight(green)`
	`  /* do something */`
	
	`  else`
	`  /* do something */`
	`  TurnOnLight(Yellow)`
	`  TakePhoto()`
	`  /* do something */`
	
	`  else`

```
                /* do something */
                <<create>> TurnOnLight(Yellow)
                TakePhoto()
                /* do something */
     else
          /* do something */
          TurnOnLight(Yellow)
          TakePhoto()
          /* do something */
```

Listing 4 Aspectual Scenario - Gizmo Validation

Since the composed scenarios already include the sequence of conditions that must hold concerning a certain goal, the sequence diagrams that represent the behavior concerning the found scenarios will be easily drawn a MATA sequence diagrams.

When representing scenarios, a "/* do something */" tag shows that something could occur in that point. This can be represented in sequence diagrams with a note marked with "REF", indicating a place where other behavior may occur.

The "Gizmo validation" correspondent sequence diagram is depicted in Figure 5-8.

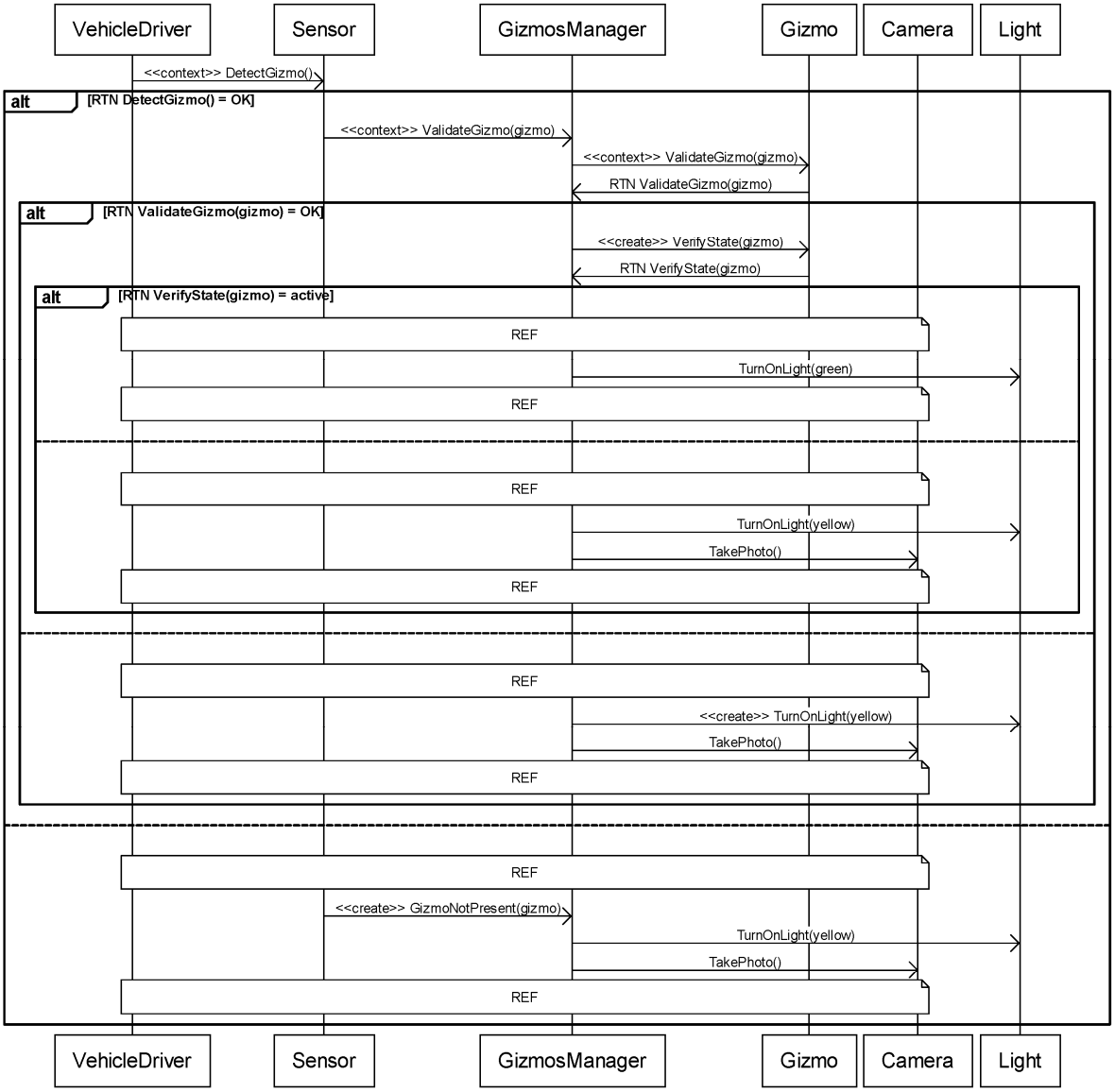

Figure 5-8 Sequence Diagram - Gizmo Validation

In what concerns the "Amoun to Pay" aspectual scenario and related sequence diagram, Listing 5 and Figure 5-9 are presented.

Amount to pay	`<<context>>(ValidateGizmo(gizmo)|| VerifyEntrance(gizmo));`
	`/* do something */`
	`<<create>> CalculateAmountToPay();`
	`/* do something */`

Listing 5 Aspectual scenario - Amount to pay

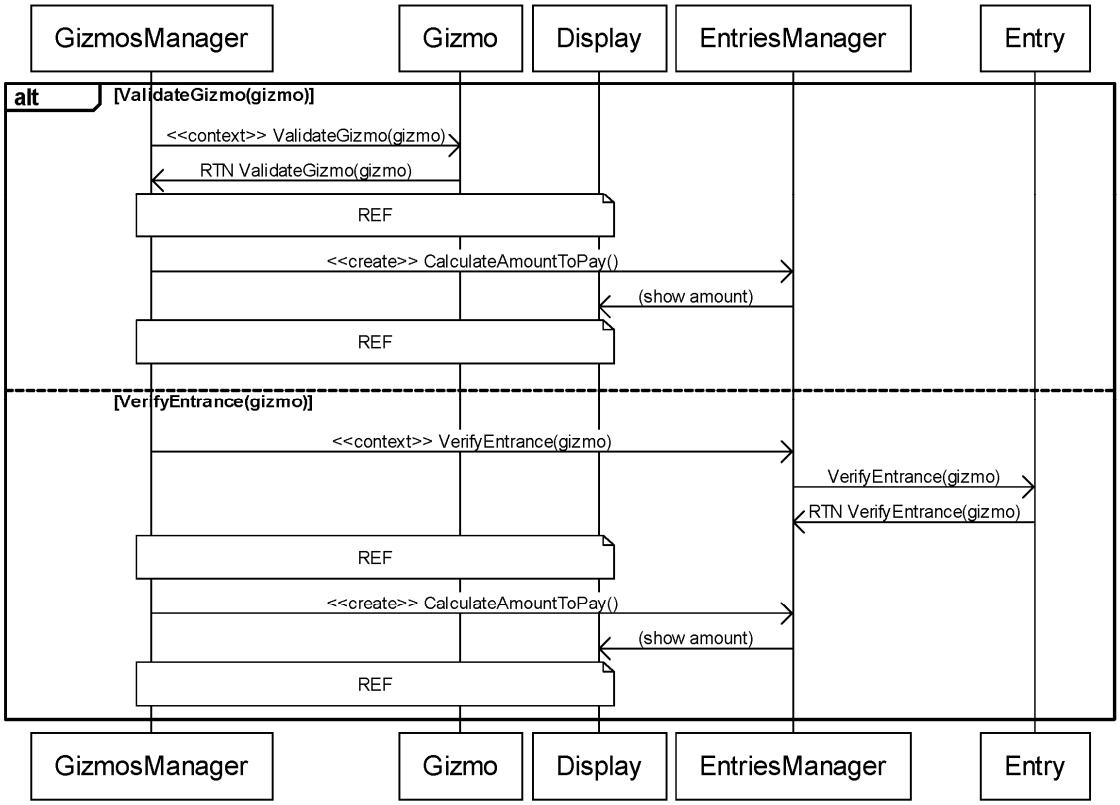

Figure 5-9 Sequence Diagram - Amount to pay

The "Exit Toll Gate" base scenario is described in Listing 6 and represented via sequence diagram in Figure 5-10.

Exit Toll Gate	`DetectGizmo();`
	`ValidateGizmo(gizmo);`
	`VerifyEntrance(gizmo);`
	`RegisterExit(gizmo).`

Listing 6 Base Scenario - Exit Toll Gate

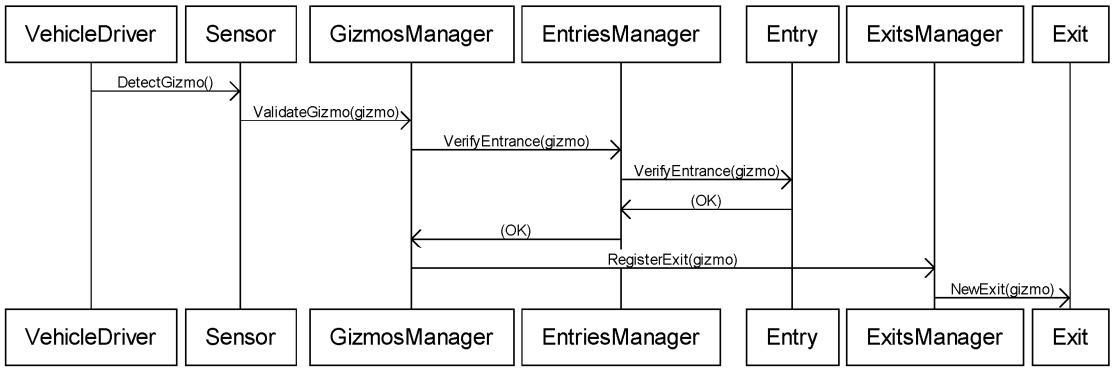

Figure 5-10 Sequence diagram - Exit Toll Gate

Finally, a description of the "Pass Single Toll" scenario and correspondent sequence diagram can be seen in Figures A and B, respectively.

Pass Single Toll	DetectGizmo(); ValidateGizmo(gizmo); RegisterPassage().

Listing 7 Base Scenario - Pass Single Toll

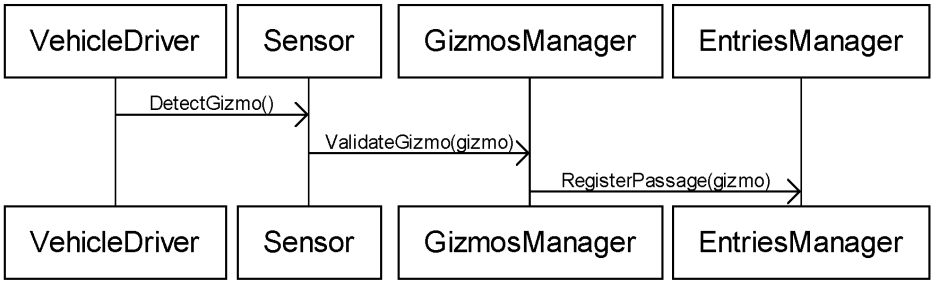

Figure 5-11 Sequence diagram - Pass Single Toll

The composed scenario regarding the "Exit Toll Gate" behavior is depicted in Listing 8 and the correspondent sequence diagram is shown in Figure 5-12.

Exit Toll Gate	DetectGizmo();
	If(RTN DetectGizmo() == OK)
	ValidateGizmo(gizmo);
	if(RTN ValidateGizmo(gizmo) == OK)
	VerifyState(gizmo);
	if (RTN VerifyState(gizmo) == active)
	VerifyEntrance(gizmo);
	if (RTN VerifyEntrance(gizmo) == OK)
	CalculateAmountToPay();
	RegisterExit(gizmo);
	TurnOnLight(green)
	else
	TurnOnLight(yellow)
	TakePhoto()
	else
	TurnOnLight(yellow)
	TakePhoto().
	else
	TurnOnLight(yellow)
	TakePhoto()
	else
	TurnOnLight(yellow)
	TakePhoto().

Listing 8 Composed Scenario - Exit Toll Gate

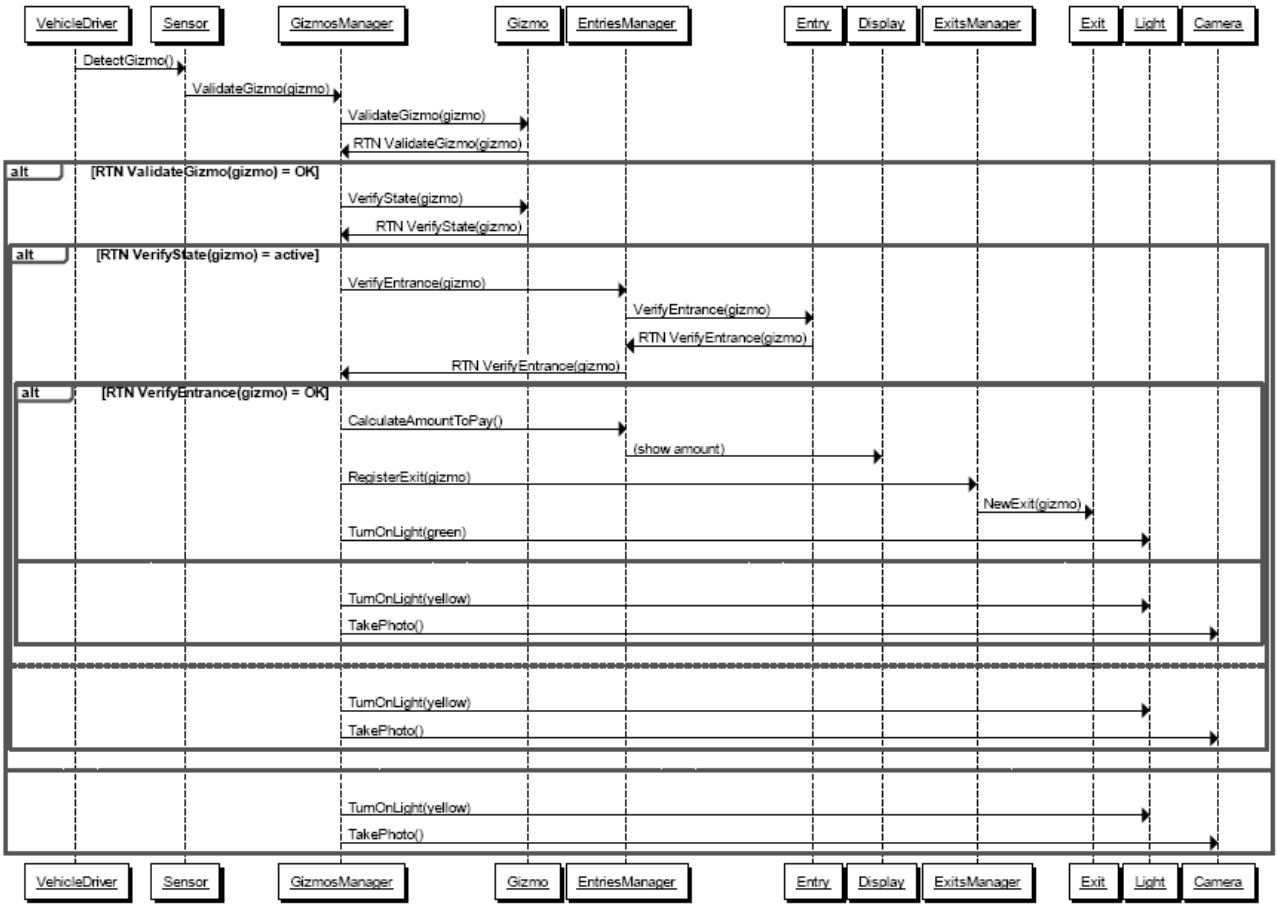

Figure 5-12 Sequence Diagram - Composed Scenario - Exit Toll Gate

Finally, a description of the "Pass single toll" composed scenario is shown in Listing 9.

Pass Single Toll	`DetectGizmo();`
	`If(RTN DetectGizmo() == OK)`
	`  ValidateGizmo(gizmo);`
	`  if(RTN ValidateGizmo(gizmo) == OK)`
	`  VerifyState(gizmo);`
	`  if (RTN VerifyState(gizmo) == active`
	`  **CalculateAmountToPay();**`
	`  TurnOnLight(green)`
	`  **RegisterPassage().**`
	`  else`
	`  TurnOnLight(yellow)`
	`  TakePhoto().`
	`  else`
	`  TurnOnLight(yellow)`
	`  TakePhoto()`
	`Else`
	`  TurnOnLight(yellow)`
	`  TakePhoto()`

Listing 9 Composed Scenario - Pass Single Toll

Figure 5-13 presents the composed behavior concerning the "Pass single toll" sequence diagram.

The remaining scenarios and correspondent sequence diagrams can be found in Annex A.

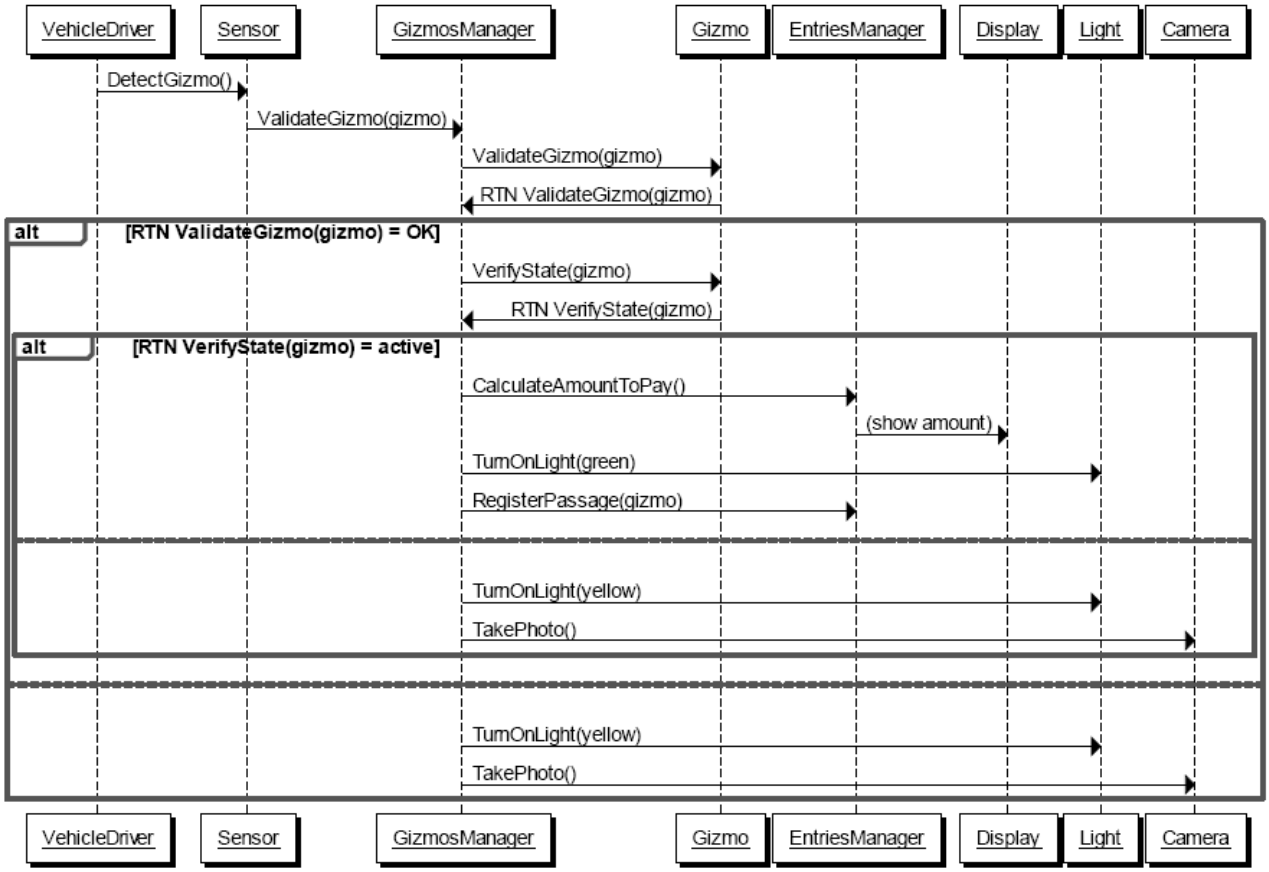

Figure 5-13 Sequence Diagram - Composed Scenario - Pass Single Toll

5.2 *Conceptual Model*

MAST (Modeling Aspectual Scenarios with Theme), beginning with a set of requirements, produces a set of themes, which are later classified as aspectual or bases, taking advantage of a set of heuristics provided by the Theme Approach to find crosscutting behavior. Each identified theme will correspond to an aspectual or base use case, respectively. A theme can also identify unexpected behavior, thus being classified as a *mistheme*.

Each use case, being seen as a goal, can be related with some actor(s), which is (are) responsible for that goal achievement. A misuse case, which can be seen as a system threat is in charge of some offender user. Each use case related scenarios can be represent as messages exchange between actors or objects. Each actor or object has a specific lifeline.

A conceptual model defines a set of concepts and their relationships, describing the semantics of an organization.
As one can state, the described approach is based upon a set of concepts, thus a conceptual model that depicts how all those concepts relate to each other can be seen in Figure 5-14.

According to the proposed conceptual model, a theme has one or more requirements and identifies one use case. A theme can be classified as an *aspectual* or *base* one, therefore so can the correspondent identified use case. Also, a base theme/use case *triggers* an aspectual theme/use case, respectively. The concept of mistheme is also considered, identifying a misuse case.

Each use case may be a responsibility of some actor or not, depending if it is included or not by another use case. Also, a use case can extend or not one another and can also mitigates the behavior described by some misuse case.
Each misuse case is carried out by an offending actor and threatens one or more use cases. A misuse case can also include another misuse case.

A use case is described by a set of scenarios, which can also be classified as aspectual or base. A misuse case is described via negative scenarios.

The existing actors can exchange messages between them and a single message may belong to more than one actor. A message can also be sent or received by an object.

Each message defines interactions on lifelines, and each interaction may correspond to combined fragment, on which one can specify operators and constraints.

An actor and an object have a specific lifeline, defining the period of time during which they exist.

Figure 5-14 Conceptual Model

5.3 *Summary*

This Chapter has presented MAST (Modeling Aspectual Scenarios with Theme), providing an overview through the whole process and using a case study as an example.
MAST has three main activities; starting with a set of requirements:

1. Theme/Doc provides a systematic way to find a system's properties through a requirements specification;
    a. Themes are identified by looking for action words presented in the requirements description;
    b. These themes are refined and help on misthemes identification;
    c. The Theme Approach heuristics is applied in order to recognize crosscutting behavior, thus, identifying aspects.
2. An aspectual use case model is produced;
    a. Each identified theme/mistheme lead to a use case/misuse case;
    b. Actors and offender actors responsible for each use case/misuse case are identified;
    c. Scenarios and Negative Scenarios are described.
3. The system is composed via MATA;
    a. Having identified aspectual and non aspectual scenarios, MATA provides an expressive composition mechanism, letting one express where, in the base, the aspectual behavior should occur.

Concluding, this Chapter has described the MAST approach, proposing a conceptual model regarding the described and explained steps and illustrating all inherent concepts and their relationships.

Having defined the approach, the next Chapter will apply it to another Case Study.

Chapter 6
Case Study: Phone Features

The Case Study presented in this Chapter is an adapted version of the “Phone Features” case study presented in [7].

Mobiles work on cellular networks and contain a standard set of services that allow phones of different types to communicate with each other.
In order for the phone to work a subscription to a mobile phone operator is required; a SIM Card contains the unique subscription and authentication parameters for that costumer. Once the SIM card is inserted into the phone, the user must insert the SIM’s PIN. If the inserted PIN were the correct one, services can be accessed.
Mobile phones support a wide range of features: voice calls, short and multimedia messaging service (SMS and MMS), media player, games applications and many others, all of them organized and available through a menu.
Incoming calls are signaled with a ring and incoming SMS and/or MMS with alerts. When a SMS or MMS is received and an alert occurs, all other sounds are momentarily muted but not paused. On the other hand, when voice call is received, all other activities are paused and their stated saved, being resumed when the voice call ends.
A user can send and edit SMS and/or MMS from the SMS/MMS application and also make voice calls from within the voice call application. However, a user can only send SMS/MMS or make a voice call if it has some credit. To have credit, the user must charge its phone.
A user can also play, pause, save and exit a game session.
From the media player menu, a customer may play audio, record audio, listen to radio, memo play and memo record, as well as start, stop, pause and resume all these functions.

This case study will only focus on the explicit referred features.

6.1 *Analyzing Requirements*

The first stage consists on identifying all the requirements related to the described system. Therefore, the following requirements were identified:

- R1) A customer must subscribe to a mobile operator.
- R2) A SIM card must be inserted into the phone in order to access all phone features, available through the menu.
- R3) The phone features can be accessed through a menu.
- R4) The menu consists of several options: make a voice call, write a SMS/MMS, use the media player, play a game.
- R5) A user can scroll through the menu and select an item to start.
- R6) Ringing is used to signal an incoming voice call.
- R7) Alerts are used to signal an incoming SMS.
- R8) When alerts and rings occur, other sounds are momentarily muted but not paused.
- R9) When a voice call occurs, other activities are paused and their state saved to be resumed when the call ends.
- R10) A user can send and edit an SMS/MMS from within the SMS/MMS application.
- R11) A user can play, pause, save and exit a game session.
- R12) The media player lets the costumer to play audio, to record audio, to listen radio, to memo play and to memo record.

- R13) The media player can start, stop, pause and resume the entire media player related functions.

- R14) The phone user must insert a correct PIN in order to access all phone features, available through the menu.

- R15) The phone user must charge its phone in order to send SMS/MMS or to make voice calls.

- R16) A user can make voice calls.

6.2 *Identifying Themes*

Analyzing the described requirements, it is possible to identify a set of actions - the potential themes:

- T1) subscribe a mobile operator

- T2) insert SIM

- T3) start menu

- T4) select menu

- T5) scroll menu

- T6) signal call

- T7) signal SMS/MMS

- T8) save audio

- T9) pause audio

- T10) mute sounds
- T11) send/edit SMS/MMS
- T12) play game
- T13) save game
- T14) exit game
- T15) start media
- T16) play media
- T17) record media
- T18) listen radio
- T19) memo play
- T20) memo record
- T21) stop media
- T22) pause media
- T23) resume media
- T24) insert PIN
- T25) charge phone

A theme represents a "coherent set of behavior". Having this in mind it is possible to group themes. Themes T1 and T2 represent the necessary actions to take in order to access all the services provided on a phone, thus they can be group on a new theme called "Mobile operator subscription". Themes T3, T4 and T5 refers the menu which contains all the phone features, thus they can also be grouped on a "menu" theme. All audio related functions are referred on themes T8, T9, T10 and T11, being grouped on a new theme - "audio". A "game" theme can also contains game related themes like T13, T14 and T15. Themes T16 to T23 concerns the media player feature, thus they can also be grouped. Finally, Themes T7 and T11 can compose a "message service" theme and T6 can be renamed to "voice call". Theme T24 also defines an action that must be done in order to access all phone features, like Themes T1 and T2 which have been grouped earlier. However theme T24 could not be grouped with them: the first group defines a behavior that occurs only one time (considering that, once the SIM is inserted, the user does not remove it) in order to prepare and let the phone to be used; theme T24 specifies a behavior that could occur several times during the phone usage, every time a user turns it off and then turns it on.

At this point, all the phone related features are extracted from the requirements and identified as the following themes.

- T1) Mobile operator subscription
- T2) Menu
- T3) Audio
- T4) Game
- T5) Media player
- T6) Message service
- T7) Voice call
- T8) Insert PIN

- T9) Charge phone

The relationship view concerning this new set of themes and the identified requirements can be seen in Figure 6-1.

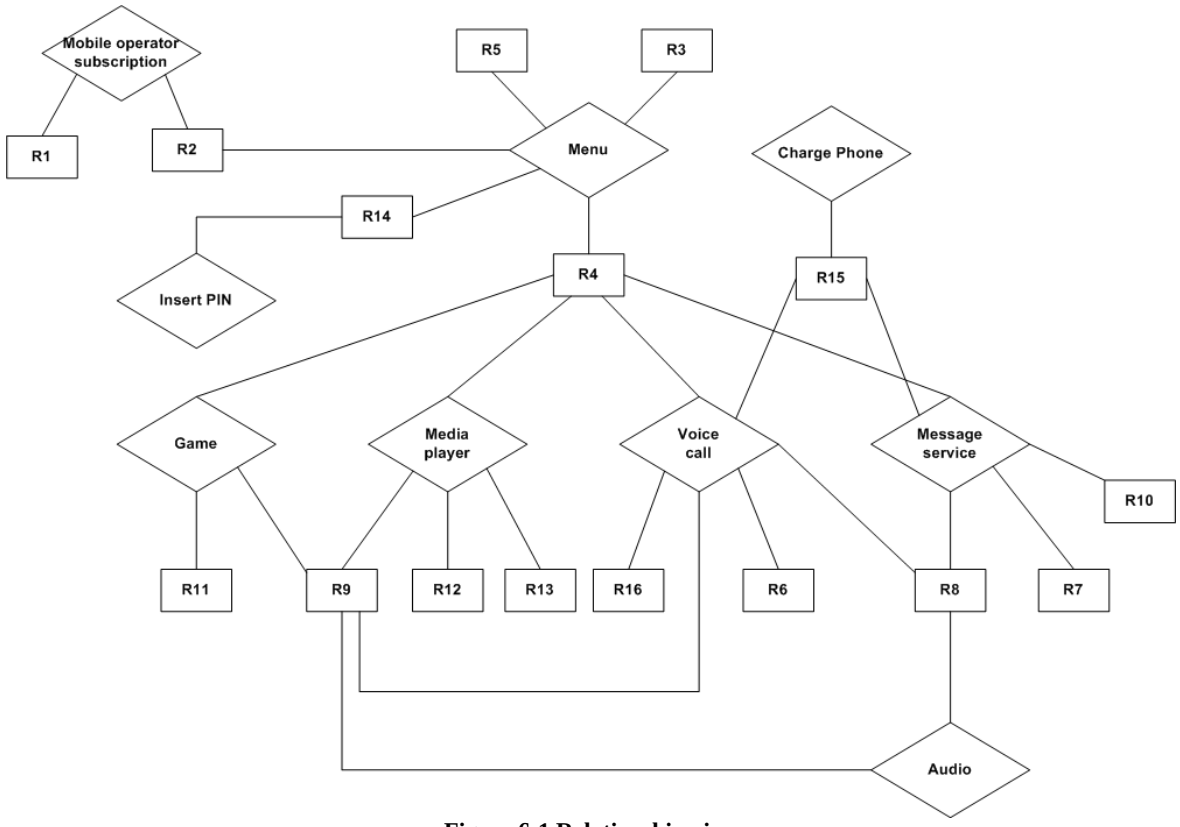

Figure 6-1 Relationship view

6.2.1 MisThemes

A system cannot be seen only from an optimist point of view. If one can predict a problematic situation, when it comes out it is possible to deal with it.

All the found themes identified a set of possible (and expected) situations. To find some related unexpected situation, we can analyze all themes and try to find something that could go wrong.

According this system's description, a user must insert the correct PIN in order to access all features. What happen if someone steals the phone and try to use it, probably inserting a wrong PIN? In that case, a message should be presented to the user, informing the illicit user that he has inserted an incorrect PIN.

In what concerns the SMS/MMS or voice call applications, an unexpected situation can easily be found: the user must have credit to do any of these actions. So, if he tries to do send a SMS/MMS or

make a voice call, even knowing that he has no credit, a message should be presented. In order to distinguish a regular user from another one that tries to deceive the system, this last will be called an “offender user”.
Thus, the following misthemes can be identified:

- MT1) Invalid PIN
- MT2) No credit

In order to deal with an unexpected situation a requirement must exist. According to the previously discussion, when each of the misthemes occurs, a message should be presented to the user, however there is no requirement, on the already defined set, that specifies such action. Hence, a negative requirement should be created.

- NR1) If there is an error, display error message to the user.

The relationship view depicted in Figure 6-1 reveals that there is some shared requirements between themes. The next step will determine to which theme a requirement should belong, identifying which of the themes related to that requirement is the dominant one.

6.3 *Identifying Crosscutting Themes*

To identify a crosscutting theme, one needs to find themes that share a requirement. There are three rules to apply on the involved themes to determine if one of the themes reveals a crosscutting one, thus as aspect:

- The requirement cannot be split up and one theme is dominant in the requirement;
- The dominant theme is triggered by other themes mentioned in the requirement, establishing a trigger relationship between the themes;
- The dominant theme is triggered in multiple situations, thus it is crosscutting and becomes the aspect.

There are five shared requirements:

- R4) The menu consists of several options: make a voice call, write a SMS/MMS, use the media player, play a game.

- R8) When alerts and rings occur, other sounds are momentarily muted but not paused.

- R9) When a voice call occurs, other activities are paused and their state saved to be resumed when the call ends.

- R14) The phone user must insert a correct PIN in order to access all phone features, available through the menu.

- R15) The phone user must charge its phone in order to send SMS/MMS or to make voice calls.

R4 is linked to "Menu", "Game", "Media Player", "Voice Call" and "Message Service".
Since the requirement describes the components of a menu there is no way to split the requirement, being clear that the "Menu" theme is the dominant one. However, the relationship between "Menu" and the other themes is not a triggering one, since there is no action that can trigger the requirement.

R8 is linked to "Audio", "Voice Call", and "Message Service" and once again cannot be split up. Analyzing the requirement definition one can see that the "Audio" theme is the dominant one, being triggered by "Voice Call" and "Message service", thus defining an aspect.

R9 is linked to "Audio", "Game", "Media Player" and "Voice call" and cannot be split up. On the other hand, the behavior described on the requirement is dominated by the "Audio" theme, also being triggered by it. The "Voice call" is also responsible for the "Audio" behavior to be triggered, which identifies "Audio" an aspect.

R14 is linked to “Menu” and “Insert PIN”, defining as a principal task to insert the PIN, thus the last theme is the one responsible for it; however there is not a triggering relationship between both referred themes – the requirement only defines something that must happen and not something is triggered by some other action.

The same goes for R15: although being linked with “Charge phone”, “Message Service” and “Voice Call”, there is not a triggering relationship between the referred themes.

All the found themes will compose the whole system, thus a connection must exist between all of them. There are three kinds of connections:

- If one theme crosscuts another one, the themes should be connected, being this kind of relationship identified with an **"<<*affects*>>"** stereotype;
- If the behavior described by another theme is required by another one in order for this theme's goal achievement, a connection must be defined using a **"<<*requires*>>"** stereotype;
- Finally, if one theme's behavior must be used by another one, a **"<<*may use*>>"** stereotype may characterize the existing connection.

Having this in mind, all the found themes and their relationships are shown in Figure 6-2.

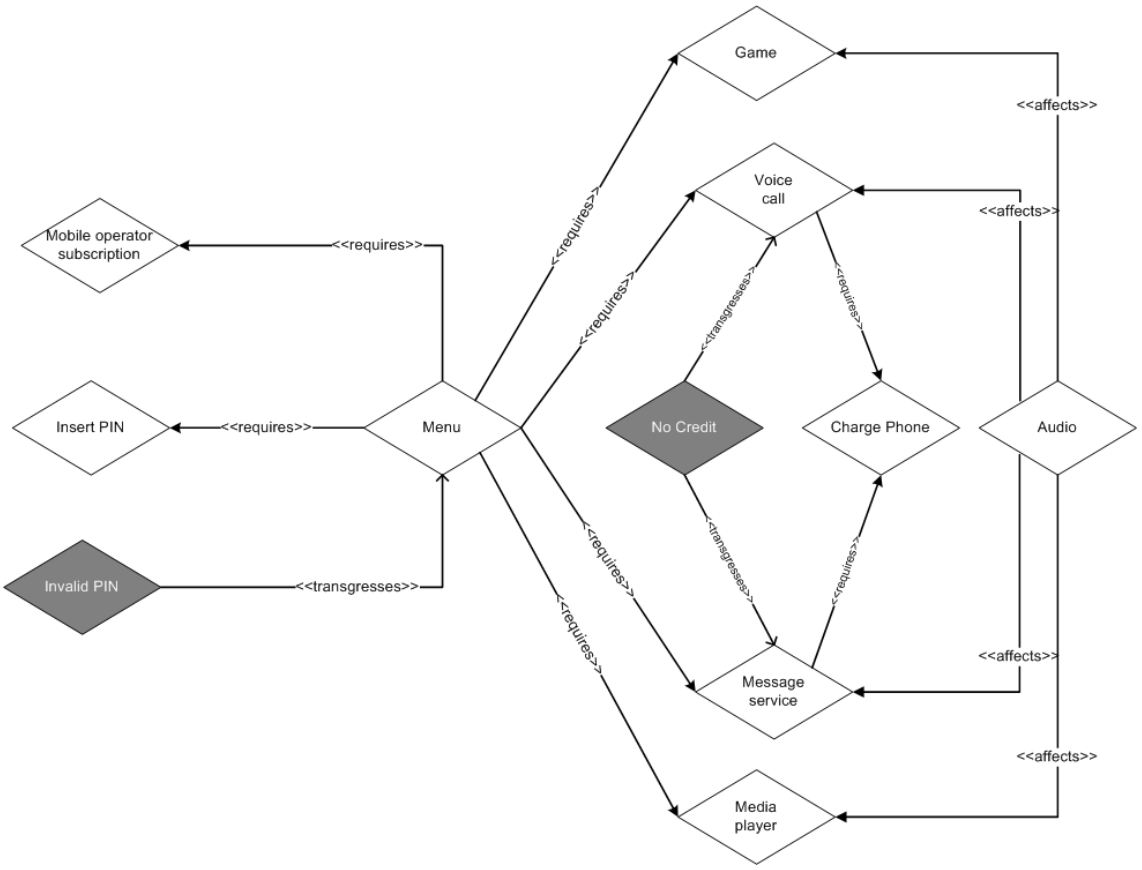

Figure 6-2 Relating Themes

6.4 *Identifying Actors, Use Cases, Misuse Cases, Scenarios and Negative Scenarios*

Each identified theme will lead to a use case. Moreover, each use case has some actor related to it. Thus, the following actors were identified:

- **Costumer:** responsible for the subscription on a mobile operator and for the SIM card to be inserted on the phone; should also insert the correct PIN and charge the phone.
- **User:** who accesses all the phone features.

Through each actor's definition and having in mind all the action identified as themes, a connection between each functionality and correspondent actor can be easily identified.

In what concerns the found misthemes, "No credit" defines a situation that is perpetuated by a specific actor – the phone user who, although knowing that has no credit, tries to deceive the system. However, in order to distinguish the user that is responsible for expected and unexpected situations, a new actor must be considered for this situation – ***Offender User***. On the other hand, an "Invalid PIN" is the result of a possible illicit user's action, thus a new actor should be considered – ***Illicit User***.

Note that there is a difference between the two identified actors: the illicit user is someone that does not have a phone, having usurped another's property; an offender user, on the other hand, is someone that has a phone but tries to trick the system.

The relationships between the identified actors and functionalities are depicted in Figure 6-3.

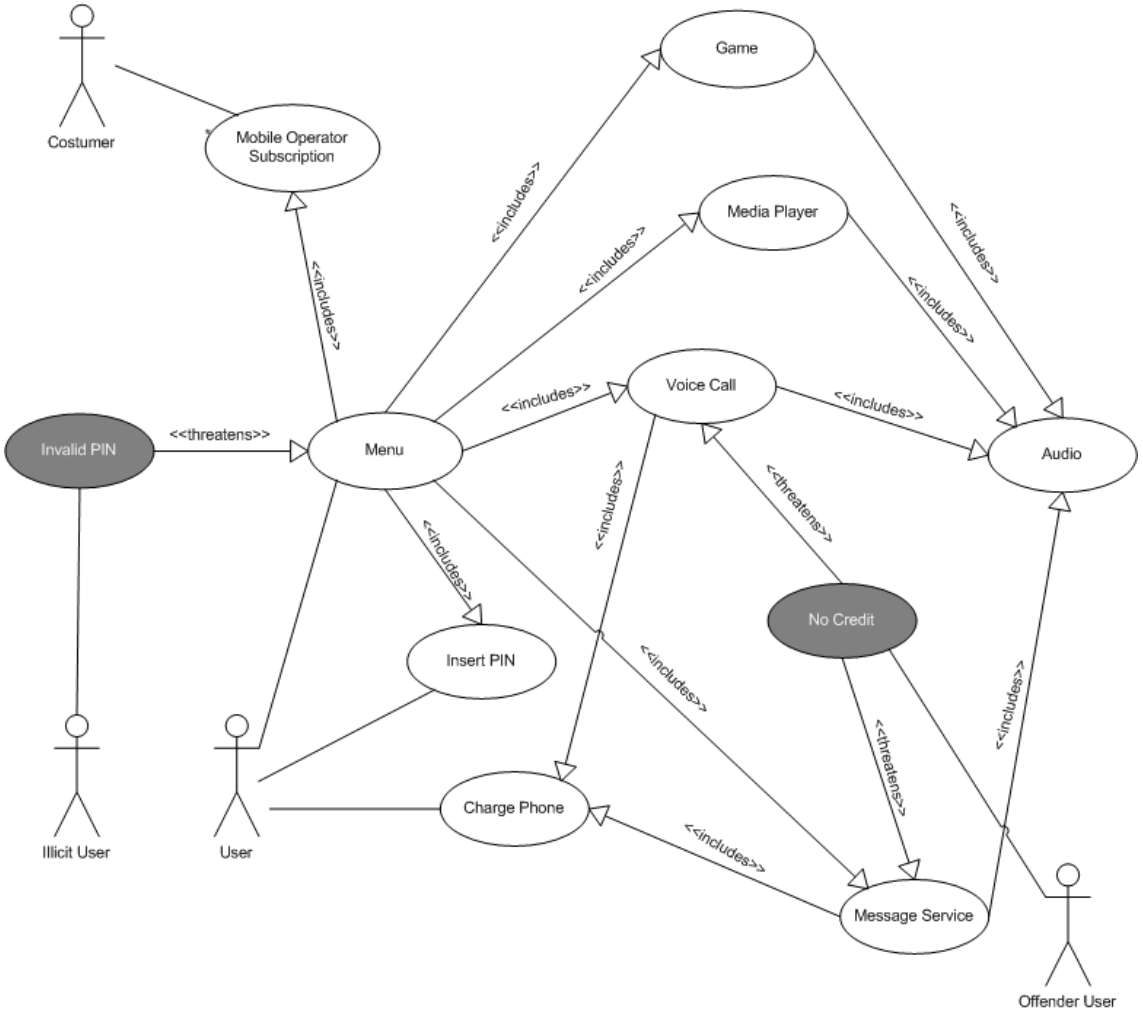

Figure 6-3 Use Cases Diagram

Since each theme results on a use case, the correspondent scenarios can be identified through the set of requirements linked to each theme/use case. Also, the negative scenarios can be found by analyzing the requirements that the correspondent misuse cases transgress.

The relationships between all the found concepts - requirements, themes and use cases - are shown in Table 4.

Table 4 Themes, Use Cases and Scenarios

Themes ←→ Use Cases	**Scenarios**
Mobile Operator	R1) A costumer must subscribe to a mobile operator.
	R2) A SIM card must be inserted into the phone in order to access all phone features, available through the menu.
Menu	R2) A SIM card must be inserted into the phone in order to access all phone features, available through the menu.
	R14) The phone user must insert a correct PIN in order to access all phone features, available through the menu.
	R3) The phone features can be accessed through a menu.

	R5) A user can scroll through the menu and select an item to start.
	R4) The menu consists of several options: make a voice call, write a SMS/MMS, use the media player, play a game.
Game	R4) The menu consists of several options: make a voice call, write a SMS/MMS, use the media player, play a game.
	R11) A user can play, pause, save and exit a game session.
	R9) When a voice call occurs, other activities are paused and their state saved to be resumed when the call ends.
Media Player	R4) The menu consists of several options: make a voice call, write a SMS/MMS, use the media player, play a game.
	R12) The media player lets the costumer to play audio, to record audio, to listen radio, to memo play and to memo record.
	R13) The media player can start, stop, pause and resume the entire media player related functions.
	R9) When a voice call occurs, other activities are paused and their state saved to be resumed when the call ends.
Voice Call	R4) The menu consists of several options: make a voice call, write a SMS/MMS, use the media player, play a game.
	R16) A user can make voice calls.
	R15) The phone user must charge its phone in order to send SMS/MMS or to make voice calls.
	R6) Ringing is used to signal an incoming voice call.
	R9) When a voice call occurs, other activities are paused and their state saved to be resumed when the call ends.
	R8) When alerts and rings occur, other sounds are momentarily muted but not paused.
Message Service	R4) The menu consists of several options: make a voice call, write a SMS/MMS, use the media player, play a game.
	R15) The phone user must charge its phone in order to send SMS/MMS or to make voice calls.
	R10) A user can send and edit an SMS/MMS from within the SMS/MMS application.
	R7) Alerts are used to signal an incoming SMS.
	R8) When alerts and rings occur, other sounds are momentarily muted but not paused.
Insert PIN	R14) The phone user must insert a correct PIN in order to access all phone features, available through the menu.
Charge Phone	R15) The phone user must charge its phone in order to send SMS/MMS or to make voice calls.
Audio	R8) When alerts and rings occur, other sounds are momentarily muted but not paused.
	R9) When a voice call occurs, other activities are paused and their state saved to be resumed when the call ends.

A description of the system's negative scenarios can be seen in Table 5.

Table 5 Misthemes, Misuse Cases and Negative Scenarios

MisThemes ←→ Misuse Cases	Negative Scenarios
Invalid PIN	NR1) If there is an error, display error message to the user.
No credit	NR1) If there is an error, display error message to the user.

6.5 *Composing the Scenarios*

The next step consists on compose all the found scenarios: base and aspectual ones.

The Theme Approach has already contributed to this step by identifying which of the themes are considered as base or aspect, which in turn has lead to base and aspectual scenarios identification. There is one crosscutting scenario: "Audio", which affects the base scenarios "Game", "Media Player", "Message Service" and "Voice Call".

The crosscutting behavior can now be composed with the non-aspectual one having in mind the behavior (base) that triggers another behavior (aspect).

The aspectual scenario can be seen in Listing 10.

Audio	<<context>> DetectSoundOrAlert()
	If(RTN DetectSoundOrAlert() == OK)
	/* do something */
	<<create>> MuteSound();
	/* do something */
	
	<<context>> DetectVoiceCall()
	If(RTN DetectVoiceCall() == OK)
	/* do something */
	<<create>>
	PauseCurrentActivities();
	/* do something */
	ResumeCurrentActivities();
	/* do something */
	Else
	/*do something*/

Listing 10 Aspectual Scenario - Audio

The sequence diagram related to the Audio behavior is depicted in Figure 2-1.

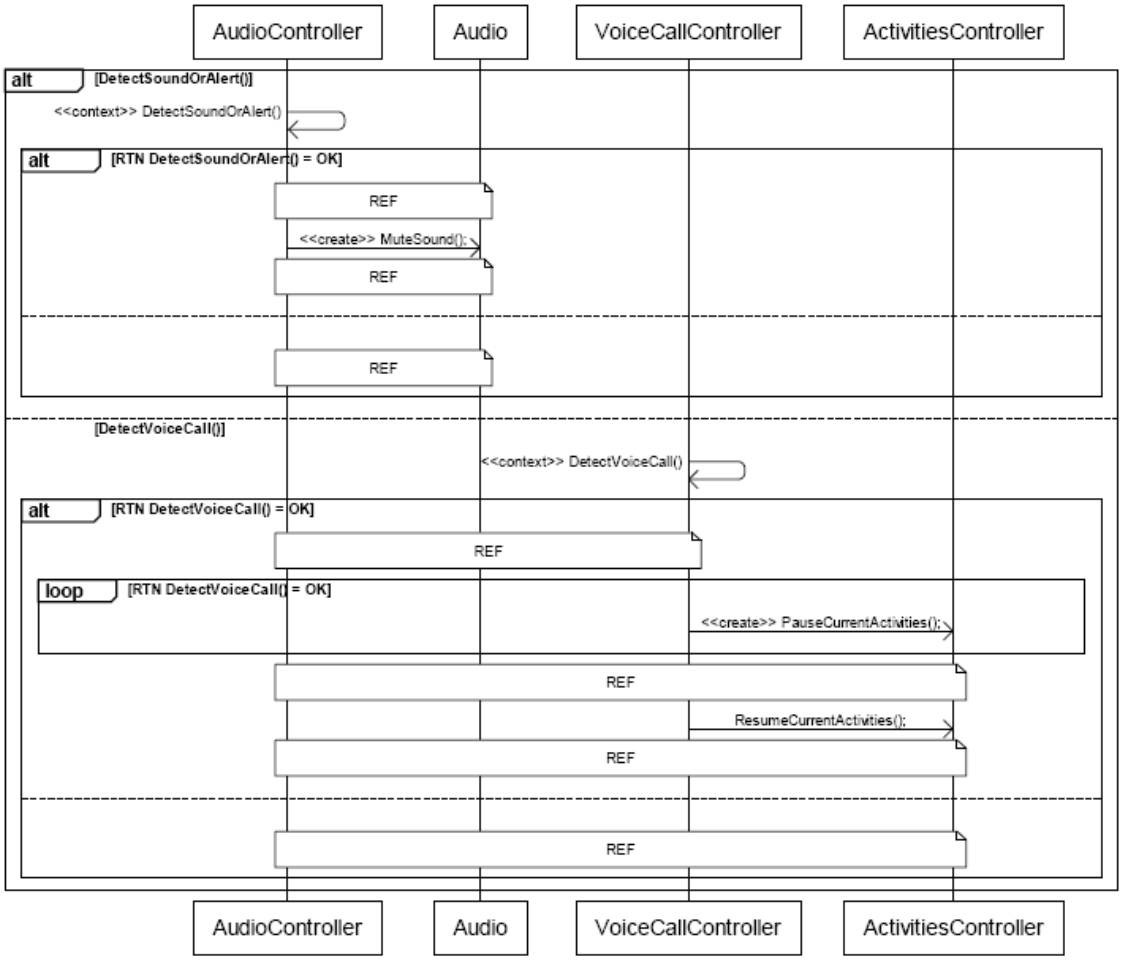

Figure 6-4 Sequence Diagram - Audio

A description of the "Game" base scenario can be seen in Listing 11.

Game	SelectGame(); DetectSoundOrAlert(); DetectVoiceCall(); ExitGame().

Listing 11 Base Scenario - Game

Figure 6-5 shows the "Game" related sequence diagram.

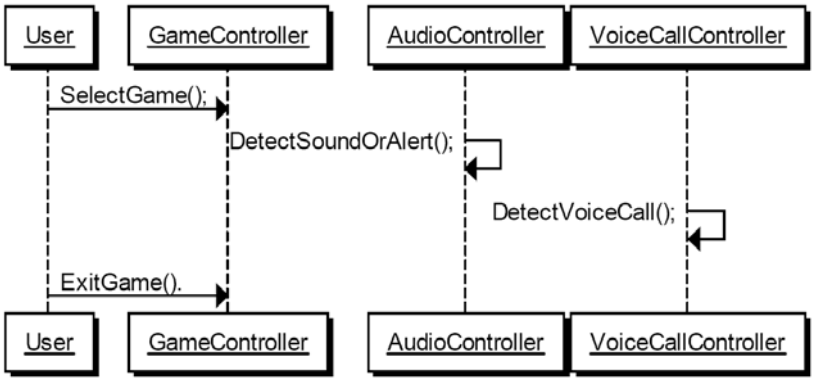

Figure 6-5 Sequence Diagram - Game

In what concerns the "Voice Call" scenario, its description can be seen in Listing 12.

Voice Call	SelectVoiceCall() || DetectVoiceCall(); If (SelectVoiceCall()) VerifyCredit() EndVoiceCall()

Listing 12 Base Scenario - Voice Call

The sequence diagram that illustrates de "Voice Call" scenario is shown in Figure 6-6.

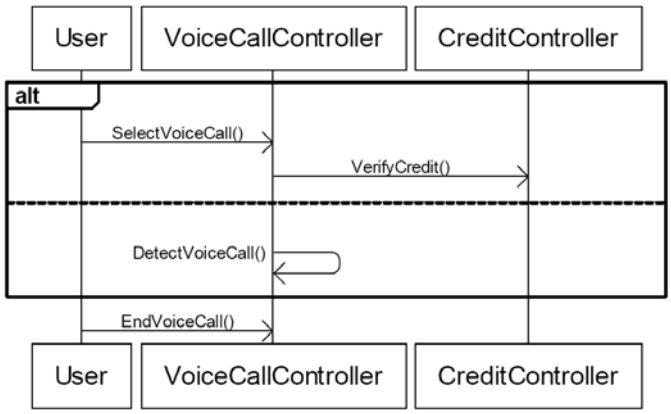

Figure 6-6 Sequence Diagram - Voice Call

A composed scenario for the "Game" can be seen in Listing 13. The correspondent sequence diagram is depicted in Figure 6-7.

Game	SelectGame(); DetectSoundOrAlert(); if(RTN DetectSoundOrAlert() == OK)

	MuteSound();
	ResumeSound()
	
	DetectVoiceCall();
	
	If(RTN DetectVoiceCall() == OK)
	PauseGame();
	SaveGame();
	EndVoiceCall();
	ResumeGame()
	else
	PlayGame() || PauseGame() || SaveGame();
	
	ExitGame()

Listing 13 Composed Scenario – Game

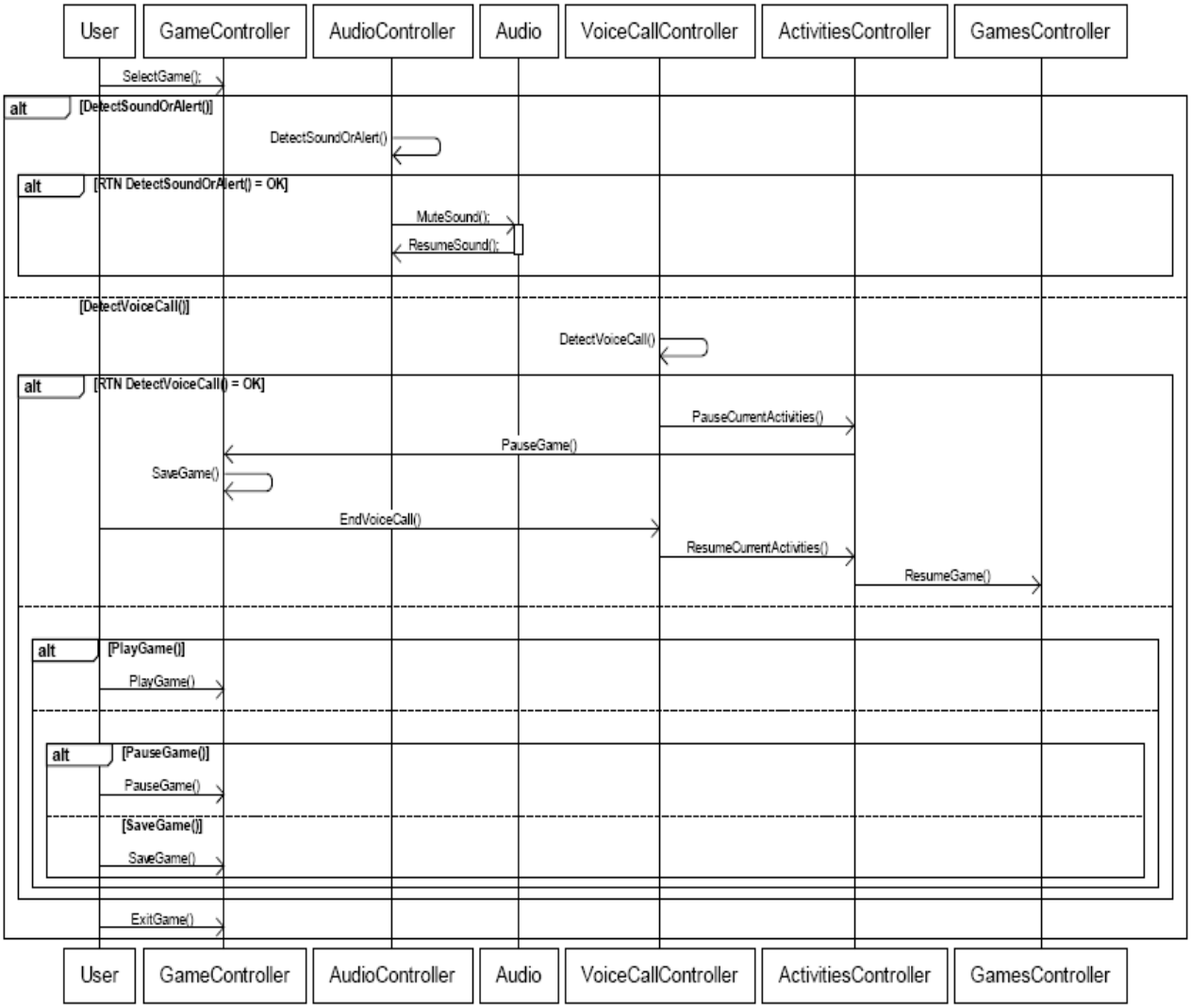

Figure 6-7 Sequence Diagram - Composed Scenario - Game

In what concerns the "Voice Call" composed behavior, its description can be found in Listing 14 and related sequence diagram in Figure 6-8.

Voice Call	If(DetectVoiceCall() == OK)
	PauseSomeActivity();
	SaveSomeActivity();
	DoVoiceCall();
	EndVoiceCall();
	ResumeSomeActivity()
	else
	VerifyCredit()
	If(VerifyCredit() == OK)
	DoVoiceCall();
	EndVoiceCall()
	Else
	DisplayErrorMessage(message)

Listing 14 Composed Scenario - Voice Call

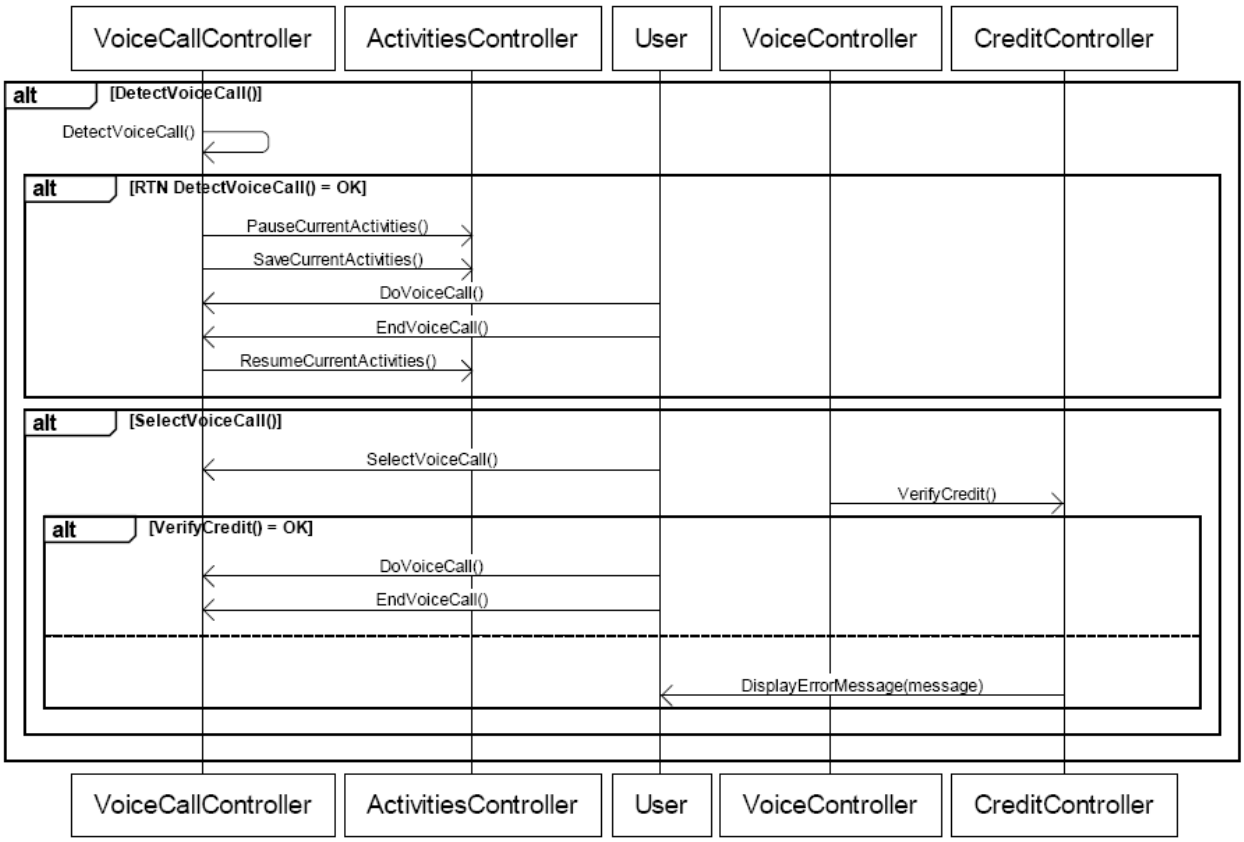

Figure 6-8 Sequence Diagram - Composed Scenario - Voice Call

The remaining base, aspectual and composed identified scenarios and correspondent sequence diagrams are depicted in Annex B.

We have identified two negative scenarios: “Invalid PIN” and “No credit”, which description can be found in Listing 15and Listing 16, respectively.

Invalid Pin	If(VerifyPIN() != OK) DisplayErrorMessage(message)

Listing 15 Negative Scenario - Invalid PIN

No credit	If (VerifyCredit() != OK) DisplayErrorMessage(message)

Listing 16 Negative Scenario - No credit

The second one has been included on the "Message Service" and "Voice Call" composed scenarios; the former has not been included, since it only happens when the user turns on the phone and is asked to insert a PIN. However, there is not any crosscutting relationship between the "Insert PIN" theme and another one, thus the occurrence of an unexpected situation has not been composed with the whole system.

Hence, and despite of not defining a crosscutting relationship, and thus, not specifying a composition between a base and aspectual behavior, the following composed unexpected scenario (Listing 17) must also be considered.

Invalid PIN	InsertPIN()
	If(VerifyPIN() == OK)
	MakePhoneFunctionalitiesAvailable()
	Else
	DisplayErrorMessage(message)

Listing 17 Composed unexpected Scenario

The correspondent sequence diagram is depicted in Figure 6-9.

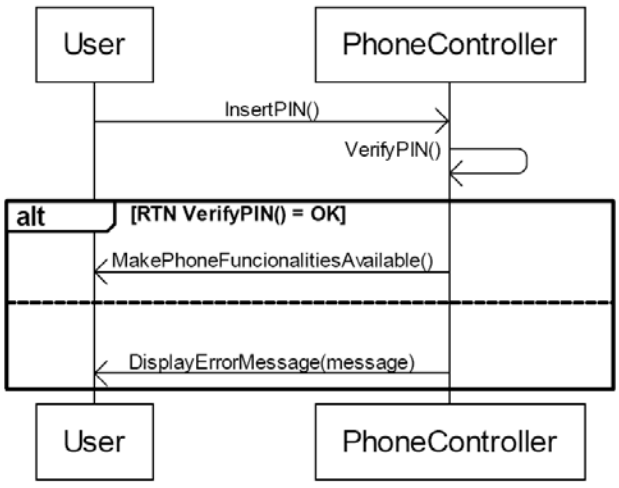

Figure 6-9 Sequence Diagram - Invalid PIN

6.6 *Summary*

In this Chapter, MAST was applied on the "Phone Features" case study.

Although the proposed approach has been introduced with a case study, the application to another case study, besides showing that the same steps are applicable in different systems, can serve to detect particular circumstances of each system.

For instance, in Chapter 5, all negative scenarios were related to some aspectual scenario, thus, when composing base with aspectual scenarios, they were automatically included. On the other hand, the behavior described by one of the discovered negative scenarios on the "Phone Features" approach, namely the "Invalid PIN", only occurred in a specific situation – when the user turns on the phone, which does not identify an aspect.
This particular case has showed that, when talking about composition, not only the base and aspectual scenarios should be considered – the impact of negative scenarios should also be analyzed, possibly producing new composed scenarios that do not necessarily describe crosscutting behavior.

Also, the application to a case study shows that the MAST composition mechanism is more expressive and "easy to read" that the one that Theme/UML provides. As an example of such composition, see Annex C.

Chapter 7
Evaluation – Comparison with the used approaches

MAST is based on the Theme Approach and takes advantage of two other approaches: Scenarios and MATA. This chapter intends to compare all the approaches in order to better evaluate the proposed approach relevance.

Each of these approaches provides some value to the Requirements Engineering field and in particular on the Aspect-Oriented one, defining and specifying concepts that help the analyst on its work.

Thus, the criterion to be applied is to check if the concepts provided by each approach are also present on MAST and vice-versa.

The six select criteria are listed below:

1. *Modeling crosscutting concerns:* evaluates if an approach supports modeling crosscutting concerns;
2. *Composition mechanism*: evaluates if an approach has any kind of composition mechanism;
3. *Aspects identification:* evaluates if an approach provides a set of heuristics to find aspects;
4. *Functionalities identification:* evaluates if an approach provides a systematic way to identify a system's functionalities;
5. *Identifying unexpected behavior:* evaluates if an approach has the capability to identify unexpected behavior.
6. *Modeling unexpected behavior:* evaluates if an approach supports modeling unexpected behavior.

These criteria were chosen taking in account all the different contribution of each intervenient approach.
Theme, an aspect-oriented approach, provides a systematic way to find and modeling crosscutting concerns and also specifies a composition mechanism in order to compose the whole system.
Functionalities identification is also an important issue, since both theme and use cases can be seen as a system's functionality.
However, and because not only the expected happen, it is also important to identify and model unexpected behavior, increasing a system integrity.

7.1 Applying the criteria

The comparison to be done will not only consider if a criterion is or is not present; it will be also specified how the criterion was instantiated on a particular approach.

Analyzing the comparison table (Table 6), one can conclude that MAST is the only approach that addresses all defined criteria. Also, through the specification of how an approach instantiates certain criterion, it is possible to see which approach has contributed to MAST on each defined criterion.

7.2 Summary

This Chapter intended to evaluate the proposed approach analyzing the concepts which are likely to be modeled. Hence, the concepts provided by each intervenient approach were listed and their visibilities on MAST were questioned. Hence, it was possible to conclude that MAST, besides providing each of those concepts, also withdraws benefits from their collective application.
Having analyzed how each approach has contributed to MAST, some conclusions can now be withdrawn.

Table 6 Comparison table

Criterion /Approach	Theme		MATA		Scenario-based		MAST	
	Present ?	How?	Present?	How?	Present?	How?	Present?	How?
1	Yes	Provides a crosscutting relationship view.	No		No		Yes	Relating themes, base and aspectual ones.
2	Yes	Through the use of arrows, templates; operator: before, around and after.	Yes	Via model transformations.	No		Yes	Via model transformation.
3	Yes	Provides a set of heuristics.	No		No		Yes	Applying the Theme Approach heuristics.
4	Yes	Identifying themes through action words identification.	No		No		Yes	Identifying themes through action words identification.
5	No		No		Yes	Through misuse cases.	Yes	Through misthemes identification.
6	No		No		Yes	Through misuse cases diagrams.	Yes	Relating themes and misthemes and use cases and misuse cases.

Chapter 8
Conclusions

With time, hardware and software complexity has increased, leading to maintenance problems. The object-oriented paradigm has attempted to solve this issue by introducing units of logic and by emphasizing reusability in software. Each independent unit, called *object*, constitutes the basis for several techniques that have contributed to software development such as *encapsulation, modularity, polymorphism and inheritance.* However, it is not always possible to modularize all the system's functionalities: pieces of code will eventually become repeated in different places, defining a phenomenon known as *scattering*, making it difficult to maintain and to evolve. Oh the other hand, each module can even contain code relating to many concerns, leading to a phenomenon called *tangling*.

Aspect-oriented programming (AOP) [14] has emerged as a solution to the scattering and tangling problems; it attempts to aid programmers in the separation of concerns, specially crosscutting ones, as an advance in modularization. Crosscutting concerns are concerns that do not get properly encapsulated in their own modules, which increase the system's complexity and makes evolution more difficult. Separation of concerns entails breaking down a program into distinct parts that overlap in functionality as little as possible. Aspect-oriented programming proposed solution consists on allowing a programmer to express crosscutting concerns in standalone modules called aspects – thus, a developer can specify behavior that overlays an existing class model.

In this dissertation, Theme was chosen as the aspect-oriented technique due to its capability to organize information pertaining to a system's requirements. A Theme can be seen as an action performed by the system, therefore all the system related themes can be found via analysis of the requirements documentation, finding all the potential actions concerning the described system. Also, this approach provides a set of heuristics that are quite useful to recognize crosscutting behavior, i.e., aspects. However, the composition mechanism at scenario level that was proposed for this approach (Theme/UML) is not expressive enough, not allowing more complex kinds of composition.

To achieve a more expressive composition, MATA (Modeling Aspects using a Transformation Approach), a scenario-based approach based on UML, was considered. The composition between base and aspectual behavior is achieved by means of graph transformations that specify where, in the base, the aspect should occur.
By integrating the Theme Approach with MATA, the previously referred lacks of the first are surpassed, resulting in a more intuitive and comprehensible vision of the whole system.
Use cases, a popular technique to describe a system requirements, has not a systematic way to find them. On the other hand, the Theme Approach [7] provides a set of heuristics that can be used to easily identify a set of functionalities regarding some system, including the ones that can be classified as aspects. Therefore, the Theme Approach emerges as a solution to find those functionalities, surpassing what is missing on scenario-based approaches.

This dissertation intends to take the advantage of three existing approaches, extracting the best of them. The Theme Approach provides a set of heuristics to find crosscutting behavior and provides a systematic way to identify use cases, a popular and easy to understand technique. However, in what concerns the composition of all the found functionalities, the existing methodology is not easy to understand nor even lets the analyst to express the desired behavior, due the limited number of available composition operators, namely AspectJ based ones like “before”, “after” and “around”. MATA [24], constituting a modeling tool that considers composition as a model transformation, arises as a perfect choice to fill the gap left by the Theme Approach.

However, not only the expected situations do happen – an unexpected situation can also occur and the system should be capable to handle it. In this context, a new concept were introduced, extending the set of concepts provided by the Theme Approach – a ***mistheme*** specifies some unexpected behavior and could be identified by questioning the found themes and trying to find some problematic situations. If the initial set of requirements does not contemplate the found issues, then new “negative requirements” are created, thus assuring the system’s integrity.

“Aspect-Oriented Analysis and Design: The Theme Approach” [7] is the first and only published book on Aspect-Oriented Requirements and Analysis and has provided a huge contribution to this work.

8.1 *Contributions*

The commonly approaches view a system from an optimist perspective: only what should happen is taking in account, whereas the unexpected it not even considered.

The approach described in this dissertation aims at identifying and modeling not only the expected situations but also the unexpected ones. By considering both optimist and pessimist perspectives, we can cover all system's boundaries in what concerns its integrity. The more we can predict, the more we can handle.

Concluding, the proposed synergy between two complementary techniques, the Theme approach and scenario-based approaches, takes the advantage of already existing approaches, surpassing its lacks and resulting on a new approach that is easy to understand, to develop and to compose.

8.2 *Future Work*

In what concerns future work, there are some points in this current work that could be improved. Namely, to complement the proposed conceptual model with OCL rules [23] in order to better specify the relationships between all concepts.

Also, a tool could be implemented: in that tool, one could identify potential themes through requirements analysis, since they would be written following a certain syntax, thus making it possible to identify the "action" to which each one refers. Also, by analyzing each theme definition (taking in account the related requirements) the relationships between themes could be deducted: a theme is related to one another if its behavior is required or mentioned by another, thus if a theme's action is referred in another theme description, then the two themes must be linked. Seeing that there is a set of well defined heuristics to identify aspect in themes, these heuristics could be also integrated in that tool in order to recognize aspectual behavior.

Finally, a Domain Specific Language [22] could also be defined, exploiting domain specific features and improving usability and comprehensibility – a DSL would then express this dissertation's approach in a formal manner. That DSL can be used as a basis for the tool implementation.

References

[1] Ian Alexander. *Scenarios, Stories, Use Cases - Through the Systems Development Life-Cycle*. Wiley, New York, 2004.

[2] Daniel Amyot. Introduction to the user requirements notation: learning by example. *Comput. Netw.*, 42(3):285–301, 2003.

[3] J. Araújo, Elisa Baniassad, Paul Clements, A. Moreira, Awais Rashid, and Bedir Tekinerdogan. Early aspects: The current landscape. Technical Report CMU/SEI-2005-TN-xxx, Carnegie Mellon University, 2005.

[4] J. Araújo and A. Moreira. An aspectual use case driven approach. In *VIII Jornadas de Ingeniería de Software y Bases de Datos*. Thompson, 11 2003.

[5] João Araújo, Ana Moreira, Isabel Brito, and Awais Rashid. Aspect-oriented requirements with UML. In Mohamed Kandé, Omar Aldawud, Grady Booch, and Bill Harrison, editors, *Workshop on Aspect-Oriented Modeling with UML*, 2002.

[6] Isabel Brito. *Aspect-Oriented Requirements Engineering*. PhD thesis, Escola Superior de Tecnologia e Gestão, Instituto Politécnico de Beja, Beja, Portugal, 2004.

[7] Siobhàn Clarke and Elisa Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach*. Addison-Wesley Professional, 2005.

[8] Paulo Coutinho and J. Araújo. Identifying aspectual use cases using a viewpoint-oriented requirements method. Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design, Workshop of the 2nd International Conference on Aspect-Oriented Software Development, Boston, USA, 17 March 2003., 03 2003.

[9] Brinksma E. (ed). Information processing systems – open systems interconnection – lotos – a formal description technique based on the temporal ordering of observational behaviour. ISO 8807, 1988.

[10] Robert B. France, Dae-Kyoo Kim, Sudipto Ghosh, and Eunjee Song. A uml-based pattern specification technique. *IEEE Trans. Softw. Eng.*, 30(3):193–206, 2004.

[11] John C. Grundy. Aspect-oriented requirements engineering for component-based software systems. In *RE '99: Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, pages 84–91, Washington, DC, USA, 1999. IEEE Computer Society.

[12] Reiko Heckel, Jochen Malte Küster, and Gabriele Taentzer. Confluence of typed attributed graph transformation systems. In *ICGT '02: Proceedings of the First International Conference on Graph Transformation*, pages 161–176, London, UK, 2002. Springer-Verlag.

[13] Ivar Jacobson and Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2004.

[14] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[15] Ana Moreira, Awais Rashid, and Joao Araujo. Multi-dimensional separation of concerns in requirements engineering. In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 285–296, Washington, DC, USA, 2005. IEEE Computer Society.

[16] Ana M. D. Moreira, João Araújo, and Awais Rashid. A concern-oriented requirements engineering model. In Oscar Pastor and João Falcão e Cunha, editors, *CAiSE*, volume 3520 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.

[17] Gunter Mussbacher, Daniel Amyot, and Michael Weiss. Visualizing aspect-oriented requirements scenarios with use case maps. *rev*, 0:4, 2006.

[18] Awais Rashid, A. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *AOSD 2003*. ACM Press, 03 2003.

[19] Awais Rashid, Pete Sawyer, A. Moreira, and J. Araújo. Early aspects: a model for aspect-oriented requirements engineering. In *Requirements Engineering 2002 (RE'02)*. IEEE Computer Society, 09 2002.

[20] I. Sommerville and P. Sawyer. Viewpoints: Principles, problems and a practical approach to requirements engineering. *Annals of Software Engineering*, 3:101–130, 1997.

[21] Geórgia Sousa, Sérgio Soares, Paulo Borba, and Jaelson Castro. Separation of crosscutting concerns from requirements to design: Adapting the use case driven approach. In *In Proc. Early Aspects Workshop at AOSD*, 2004.

[22] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35:26–36, 2000.

[23] Jos Warmer and Anneke Kleppe. *The Object Constraint Language : Precise Modeling with UML*. Addison Wesley Longman, Inc., 1999.

[24] J. Whittle and P. Jayaraman. "mata: A tool for aspect-oriented modeling based on graph transformation". Workshop on Aspect Oriented Modeling at MODELS 2007, 2007.

[25] Jon Whittle and João Araújo. Scenario modelling with aspects. *IEE Proceedings - Software*, 151(4):157–172, 2004.

[26] Jon Whittle, Richard Kwan, and Jyoti Saboo. From scenarios to code: An air traffic control case study. *Software and System Modeling*, 4(1):71–93, 2005.

Appendix A

Toll Collection System

Enter Toll Gate	DetectGizmo(); ValidateGizmo(gizmo); RegisterEntrance(gizmo).

Listing 18 Base Scenario - Enter Toll Gate

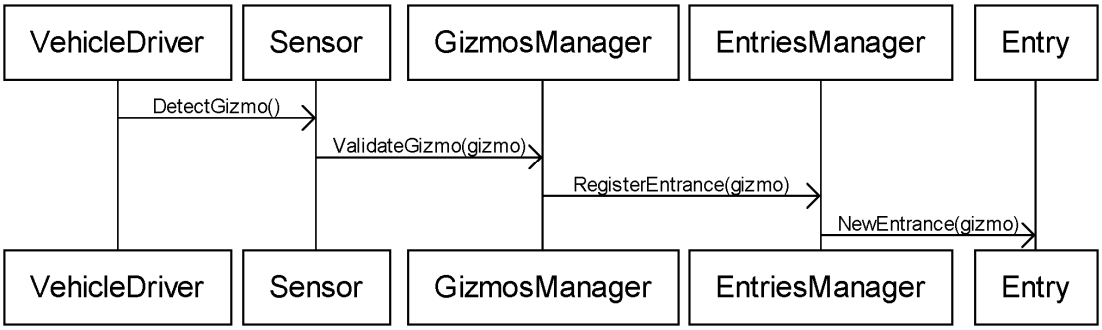

Figure 0-1 Sequence Diagram - Enter Toll Gate

Enter Toll Gate	DetectGizmo();
	If(RTN DectectGizmo() == OK)
	ValidateGizmo(gizmo);
	if(RTN ValidateGizmo(gizmo) == OK)
	VerifyState(gizmo);
	if (RTN VerifyState(gizmo) == active)
	TurnOnLight(green)
	RegisterEntrance(gizmo)
	else
	TurnOnLight(yellow)
	TakePhoto()
	else
	TurnOnLight(yellow)
	TakePhoto()
	Else
	TurnOnLight(yellow)
	TakePhoto()

Listing 19 Composed Scenario - Enter Toll gate

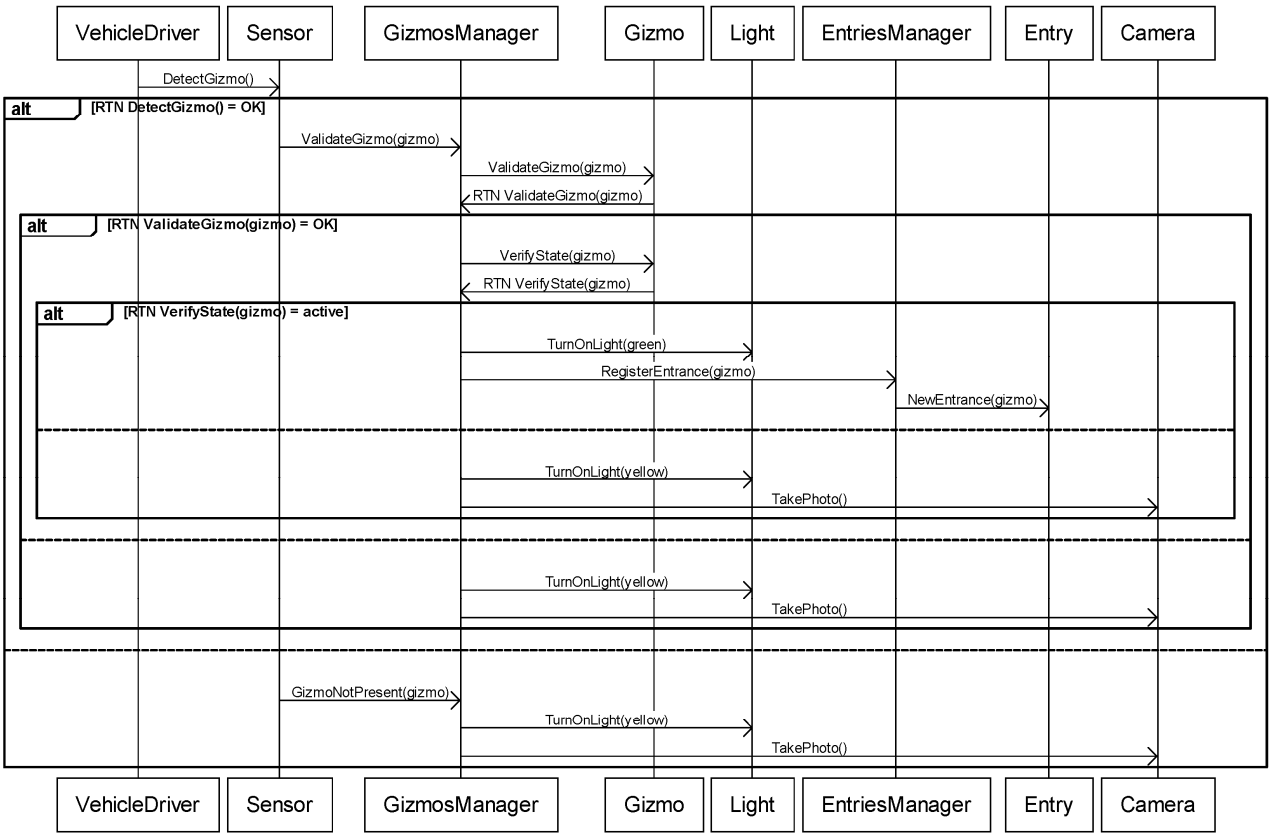

Figure 0-2 Sequence Diagram - Composed Scenario - Enter Toll Gate

Appendix B

Phone Features

Media Player	SelectMediaPlayer(); DetectSoundOrAlert(); DetectVoiceCall(); ExitMediaPlayer().

Listing 20 Base Scenario - Media Player

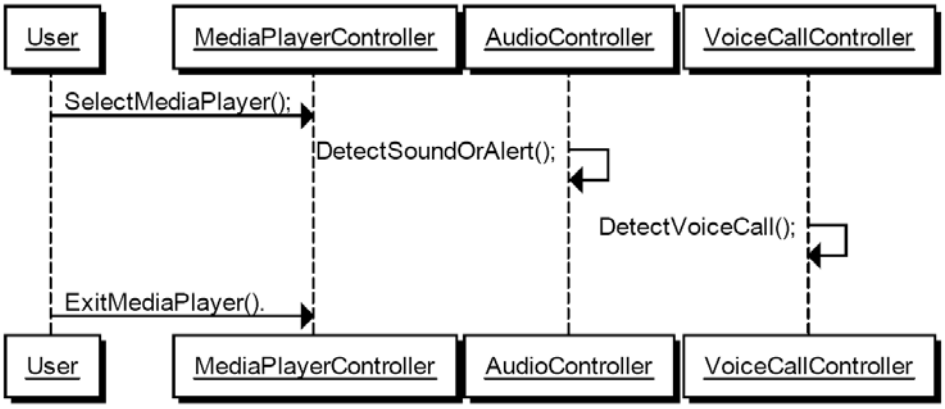

Figure 0-1 Sequence Diagram - Base Scenario - Media Player

Message Service	SelectMessageService() ||
	DetectIncomingMessage();
	If (SelectSendMessage())
	VerifyCredit()
	Else
	EditMessage()
	
	ExitMessageService().

Listing 21 Base Scenario - Message Service

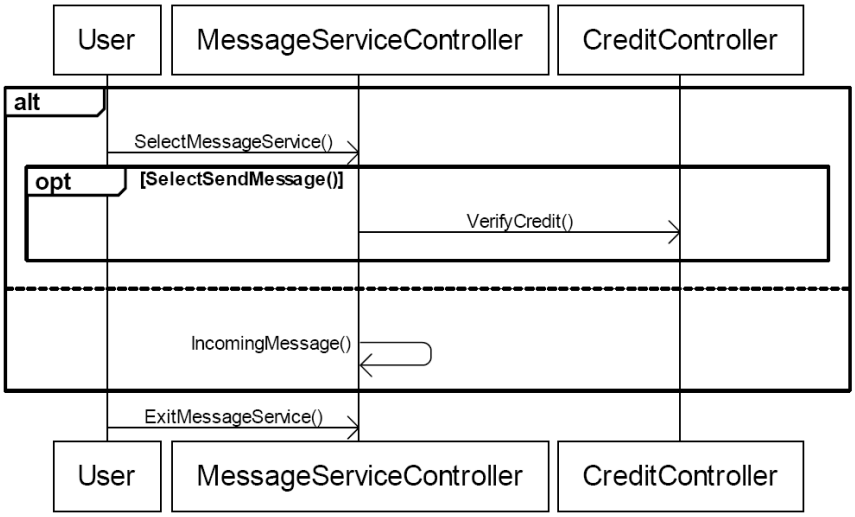

Figure 0-2 Sequence Diagram - Base Scenario - Message Service

Media Player	SelectMediaPlayer();
	DetectSoundOrAlert();
	if(RTN DetectSoundOrAlert() == OK)
	Mutesound();
	ResumeSound();
	
	DetectVoiceCall();
	
	If(RTN DetectVoiceCall() == OK)
	PauseMediaPlayerActivity();
	SaveMediaPlayerActivity();
	EndVoiceCall();
	ResumeMediaPlayerActivity();
	Else
	StartMediaPlayer() ||
	StopMediaPlayer() ||
	PauseMediaPlayer() ||
	ResumeMediaPlayer();
	
	ExitMediaPlayer().

Listing 22 Composed Scenario - Media Player

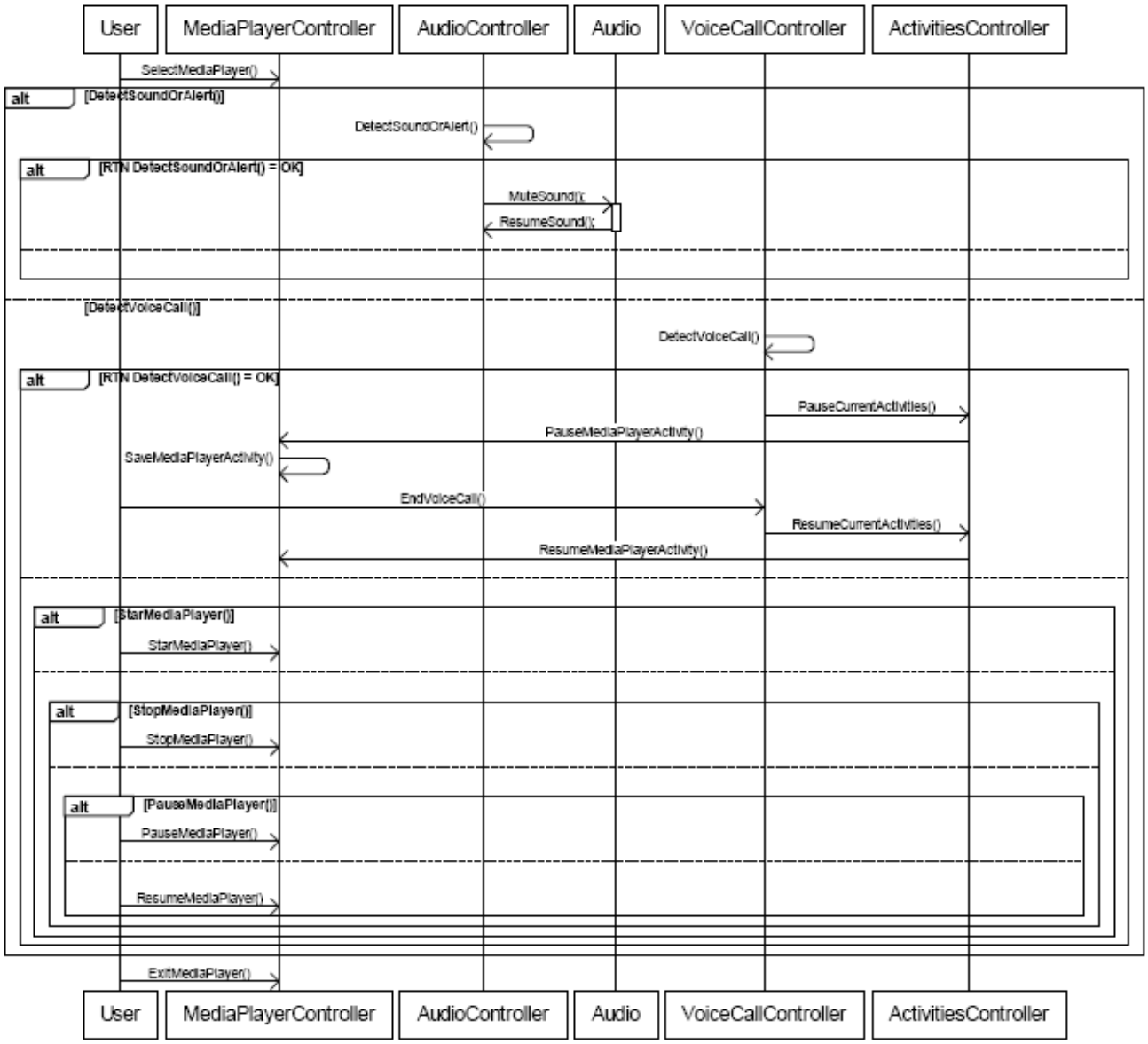

Figure 0-3 Sequence Diagram - Composed Diagram - Media Player

Message Service	If(DetectIncomingMessage () == OK)
	MuteSound();
	ReceiveMessage();
	ResumeSound()
	else
	If (SelectSendMessage())
	VerifyCredit()
	If(VerifyCredit() == OK)
	SendMessage();
	Else
	displayErrorMessage(message)
	Else
	EditMessage()
	
	ExitMessageService().

Listing 23 Composed Scenario - Message Service

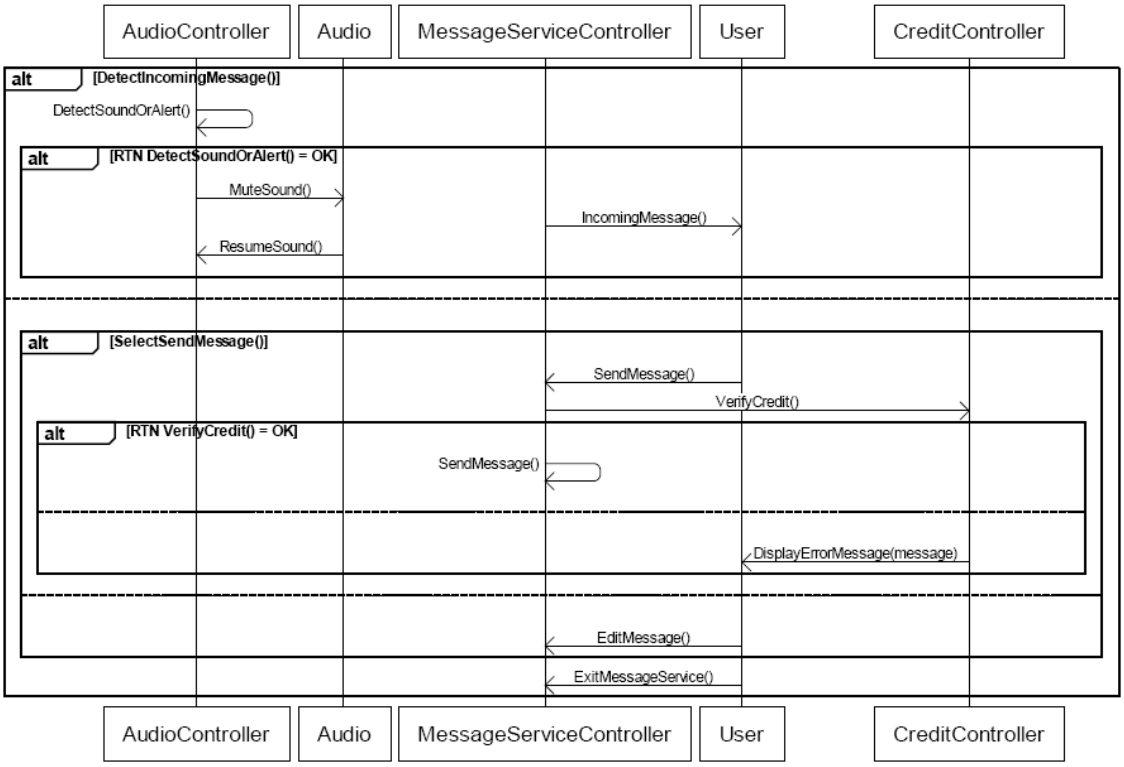

Figure 0-4 Sequence Diagram - Composed Scenario - Message Service

Appendix C

Composition on Theme Approach

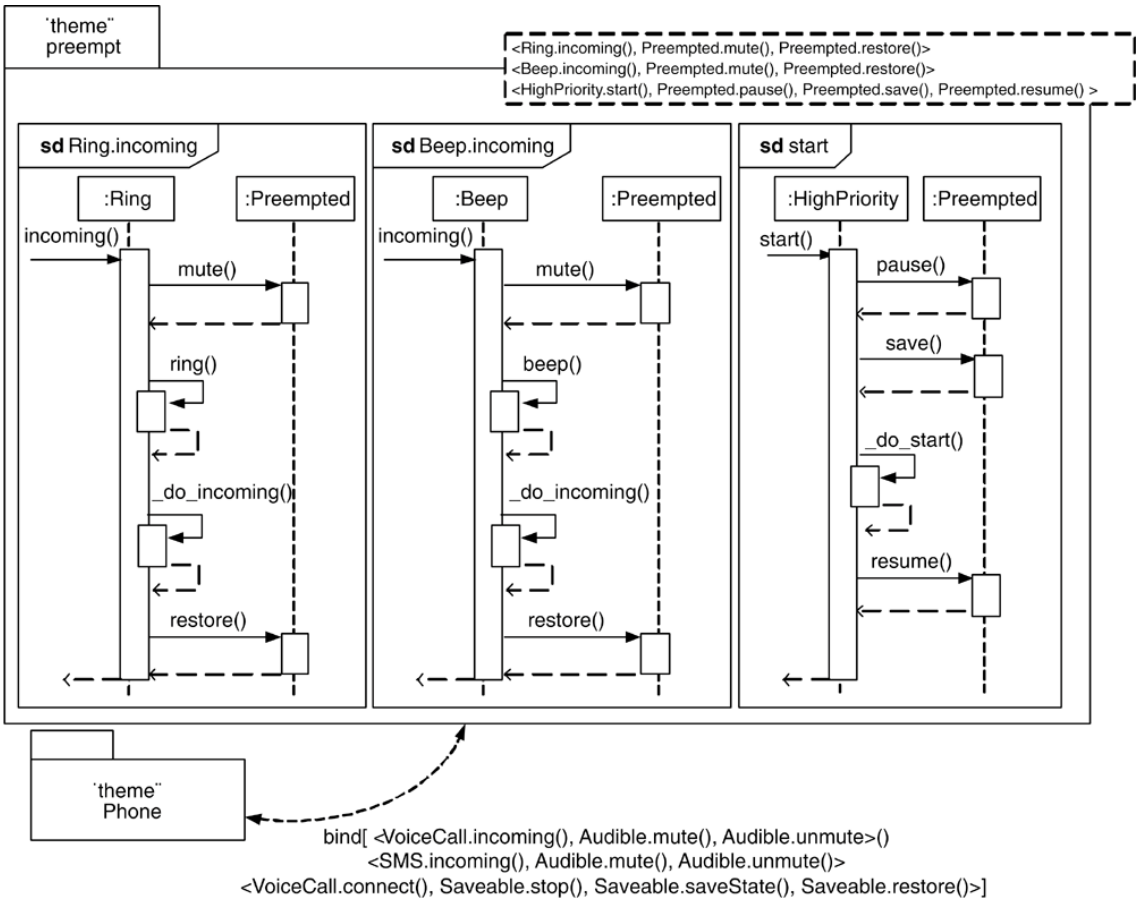

Figure 0-1 Composition on Theme Approach